AFRL-RI-RS-TR-2008-15
Final Technical Report
January 2008

REAL TIME ESTIMATION AND PREDICTION USING OPTIMISTIC SIMULATION AND CONTROL THEORY TECHNIQUES

WarpIV Technologies, Inc.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2008-15 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/ /s/

TIMOTHY E. BUSCH
Work Unit Manager

JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JAN 2008	Final	Sep 06 – Dec 07

4. TITLE AND SUBTITLE

REAL TIME ESTIMATION AND PREDICTION USING OPTIMISTIC SIMULATION AND CONTROL THEORY TECHNIQUES

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA8750-06-C-0218

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Jeffrey Steinmen
Craig Lammers

5d. PROJECT NUMBER
459S

5e. TASK NUMBER
N6

5f. WORK UNIT NUMBER
05

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
WarpIV Technologies, Inc.
5230 Carroll Canyon Road, Suite 306
San Diego CA 92121

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RISB
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-RI-RS-TR-2008-15

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 08-0094

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This paper describes a revolutionary approach to information processing that will increase warfighter productivity by orders of magnitude for problems requiring analysis, planning, optimization, and dynamic situation assessment and prediction in live operational environments. This technology, known as HyperWarpSpeed, breaks the four-dimensional barrier of space and time by extending modeling, simulation, and analysis capabilities into the fifth dimension.

15. SUBJECT TERMS
Real-Time Estimation and Prediction, Parallel Processing, Branching, HyperWarpSpeed, Kahlman Filters

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Timothy E. Busch
U	U	U	UL	44	19b. TELEPHONE NUMBER (*Include area code*) N/A

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Simulations are often used to analyze the performance of battle plans for complex scenarios. Such analytic simulations are often independently executed many times with different parameter settings to determine the optimal configuration of a complex system. Monte Carlo simulations that model critical decisions using probabilistic distributions and random number generation often require large numbers of end-to-end simulation replications to determine the statistical measures of effectiveness and/or performance of complex systems. Real-time estimation and prediction decision aid tools project the future through simulation using real-time data to calibrate the current state and require multiple-hypothesis what-if branching capabilities to simultaneously explore the effectiveness of various decision options and battle plans [1, 2, 13].

These different kinds of simulation applications are traditionally supported by running multiple executions of the same simulation using (1) different parameter settings, (2) different starting random number seeds, or (3) different critical decision options and/or battle plans. Simplistic parallel processing techniques can be used to help speed up these execution times by farming the replications to multiple resources across a computer network. However, this simple approach ignores the fact that each of the replicated simulation executions may redundantly repeat many of the same time-consuming calculations. Furthermore, final outcomes are generally provided only at the completion of the simulation runs. It is extremely challenging to consider, across all runs, the common threads of execution and relative frequencies of certain decision points and branches across all replications.

Some improvements over the brute force multiple-replication Monte Carlo approach have been explored. One example is to replicate the simulation only at branch points [3]. This has the appeal of sharing computations up to the branch point, but it does not take full advantage of computation sharing afterwards. Furthermore, simulation cloning can be cumbersome to operate and manage. Another approach clones objects at critical branch points [8]. While this offers a higher degree of computation sharing, it still does not take full advantage of the potential computation sharing for complex objects with multiple concurrent event patterns. Furthermore, neither approach merges branches to eliminate redundant event computations.

A computationally more efficient approach to support these types of simulations is to automatically detect and reuse redundant event calculations that are shared between the multiple branches at the state attribute level. Unique calculations for a particular hypothesis branch would only be performed when required. As a further optimization, this more efficient computational approach should also take advantage of parallel processing when multiple processors are available through the use of advanced optimistic event processing algorithms. Large scenarios that require considerable amounts of memory can be supported through scalable parallel and distributed algorithms that distribute simulated entities across computing resources. Multiple hypotheses branching simulation applications can be supported with minimal computational overhead on single-processor machines, multi-processor machines, and/or networks of such machines.

This report describes techniques that address all of these stated issues. First, the branching techniques for supporting parallel overlapping universes in the fifth dimension are described. Then, background on parallel processing and discrete event simulation is presented, along with an emphasis on new standards that are being promoted within the Simulation Interoperability Standards Organization (SISO). These standards are important to bring this technology to mainstream programs. A brief background is then given on real-time estimation and prediction, indicating how the combined concepts of control theory and optimistic simulation can be applied to support dynamic situation assessment and prediction. Then a discussion is provided on how to statistically analyze results generated by multiple branches using the Statistical Algebra Package that is provided within the WarpIV Kernel. Then, a framework for automatically computing measures of effectiveness and performance of a simulation is discussed. Next, a comparison is made between the approach recommended in this report and other approaches that have been used in the past. Finally, several test and performance results are provided for various hardware architectures to verify correctness of the approach and to indicate the expected processing performance of several representative real-world problems.

2 Multi-Hypothesis Branching

From a scientific context, the concept of multi-hypothesis branching has been referred to more abstractly as simulating parallel overlapping universes in the fifth dimension. The following simple illustration introduces the concepts of parallel overlapping universes in the fifth dimension.

Imagine within a simulated world that two friends, Jim and Nancy, agree to meet for lunch. Jim arrives at the restaurant first and decides to order his meal. Nancy is running late, so Jim finishes his lunch and then orders dessert. Jim now has to decide on whether to order a pricy ice cream sundae or an inexpensive chocolate chip cookie. At this point, Jim flips a coin to decide what to order. In the HyperWarpSpeed time management algorithm, Jim branches into two overlapping parallel universes. In one universe, Jim enjoys his ice cream sundae, while in the other universe Jim eats his cookie. Meanwhile, cars driving by the restaurant are unaffected by what dessert Jim has ordered. Both branches (universes) of Jim share (overlap) the modeling of the automobile traffic occurring outside the restaurant.

Nancy finally arrives as Jim is eating his dessert. Nancy asks Jim if he wouldn't mind sharing some of his dessert. At this point, Nancy automatically splits into two. In one universe, Nancy enjoys the ice cream sundae with Jim. In the other universe, Nancy shares Jim's cookie. After they finish dessert, Jim and Nancy leave the restaurant and continue on with the rest of their day. Their universes merge because the rest of their activities do not depend on what they had for dessert.

That night, Jim decides to go to a fancy restaurant for dinner. Jim really wants to order a steak but it is expensive. The Jim that had the cookie has enough money in his wallet to order the steak dinner, but the Jim in the alternate universe who ordered the ice cream sundae only has enough money for the chicken dinner. Jim automatically splits again as he orders his meal. What caused the split is the fact that the money in Jim's wallet was different depending on the path he had taken earlier in the day.

While this example assumes the rest of the world is unaffected by the couples behavior, it is entirely possible to construct a simulation in which branch points are completely independent from one another. For example, branching on whether or not a region was attacked by a nuclear weapon could affect the entire simulation in that there would be nothing in common and no possibility of overlap between the branches from that point forward.

2.1 HyperWarpSpeed

The HyperWarpSpeed branching technology was developed to (1) transparently manage large numbers of Monte Carlo replications within a single simulation execution while sharing all common computations between replications, (2) provide scalable performance on a wide variety of sequential, parallel, and distributed computing architectures, and (3) support real-time estimation and prediction [6] using live data and control theory [10] to continuously calibrate the simulation while aggressively predicting future outcomes [20].

The HyperWarpSpeed algorithm provides three significant performance benefits. *First*, redundant computations that would be performed by running independent Monte Carlo simulation replications are eliminated. If each simulation replication performed 90% of the same computations as the others, then combining the replications into a single execution would improve performance by an order of magnitude. *Second*, optimistic time management algorithms provide parallel processing capabilities on next-generation multicore computer architectures. This can provide additional orders of magnitude in speedup. *Third*, large computational savings can be achieved by continuously integrating live data into the simulation through rollback-based optimistic event processing and mathematical control theory techniques. Instead of restarting simulations as new data arrives, only the portions of the simulation that are affected by the newly received data are rolled back and recomputed (or rolled forward if possible). If restarting new simulations recalculated 90% of the same computations from previous runs, another order of magnitude performance gain would be achieved.

One can think of this approach as continually updating the three-dimensional Common Operational Picture (COP) that is embedded within the simulation using real-time estimation techniques. Predictions are continually refined because they are based on the embedded COP as it evolves over time. By providing rollback and rollforward event processing capabilities, control over the fourth dimension (i.e., time) is achieved for the embedded COP. Integrating the four-dimensional COP with branching technology provides a five-dimensional COP capability that is

able to manage and explore multiple futures. This new approach can be thought of as simulating parallel overlapping universes in the fifth dimension.

All of the capabilities described in this report are provided to the model developer and simulation operator in a manner that is robust and easy to use. The underlying complexities are completely hidden from the model developer. Providing this extremely complex capability in a simple and transparent manner to the model developer is critical for promoting its widespread use for command and control dynamic situation assessment and prediction applications.

2.1.1 The HyperWarpSpeed Time Management Algorithm

Replication numbers are used in Monte Carlo simulation to identify individual runs of the simulation with different input parameters, random number seeds, and/or planned strategies. So, 128 replications would be understood to contain the set of runs identified as replications {0, 1, … 127}.

Fundamental to understanding the HyperWarpSpeed algorithm is the notion of replication sets. Replication sets are managed as bit fields in an integer array where each bit in the array represents a particular replication. So, a replication set for 128 potential replications is stored in an array of four 32-bit integers. The array size specifies the total number of replications managed within the single execution of the HyperWarpSpeed simulation. Analysts are able to specify the size of the replication set to meet the analysis needs of the simulation run. At the start of the simulation execution, all (1) state variables and (2) initially scheduled events are identified with the full replication set.

A simple example of a replication set with 32 potential replications containing replications {0, 1, 4, 9, 15, 21, 29} would be stored as binary [0010 0000 0010 0000 1000 0010 0001 0011], where the rightmost bit represents replication 0, the next bit to the left represents replication 1, etc. High-speed bit masking operations automate critical bookkeeping for replication sets. It is critical to minimize all computational overheads when manipulating bit fields for replication sets.

Each event in the HyperWarpSpeed algorithm is associated with a replication set. At the start of the simulation, all events start out representing the full set of replications. Bayesian branching splits the replication set of the current event based on a specified or computed probability for each branch taken. The branching construct is extremely simple to use and looks like the familiar switch/case statement. The first two arguments identify labels for the two branch points. The third argument is the probability of taking the first branch. An example of this is shown in Code Segment 1.

```
BRANCH(1, 2, 0.7)
  LABEL(1)
    // Code that handles first branch
    break;

  LABEL(2)
    // Code that handles second branch
    break;

END_BRANCH
```

Code Segment 1: Code example of the branch construct placed within a user-defined event method. The probability of branching down label 1 is 0.7 and the probability of branching down label 2 is 0.3.

New multireplication primitive data types for integers, doubles, and Booleans, have been developed that internally store different values depending on how they are modified by events with different replication sets. Through operator overloading, accessing and assigning values to multireplication data types is completely transparent to the modeler. These new data types simply behave as normal integer, double, and Boolean variables. The structure for the primitive multi-replication data types is shown in Figure 1.

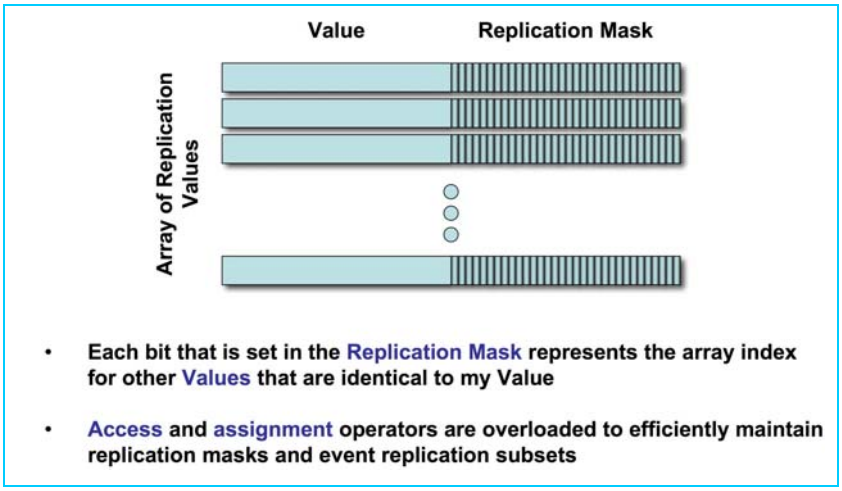

Figure 1: Multi-replication data types are represented as an array of values with a replication mask for each value. Bits set in the mask identify which other array elements have the same value. Current data types supported are integer, double, and Boolean.

Each multi-replication data type stores an array of values along with a bitfield for each value. The index of the array indicates the replication number. Each bit in the bitfield represents the set of replications that have the same value as the array element value (see Table 1).

Table 1: An example of a replicated integer with 32 replications. The value for each replication and the replication bit-mask are stored in the array. The bit-mask identifies which replications have the same value. Since the value 0 is stored in replications {0, 9, 31}, the bit mask has three 1's in the corresponding bits. The rest of the bits are set to 0.

Rep	Value	Mask (binary)	Rep	Value	Mask (binary)
0	0	1000 0000 0000 0000 0000 0010 0000 0001	16	2	0011 0001 0011 0001 1000 0000 0000 1000
1	5	0000 0000 1000 0000 0100 0001 1000 0010	17	7	0000 0000 0000 0010 0000 0000 0000 0000
2	1	0100 0000 0100 1100 0011 0000 0111 0100	18	1	0100 0000 0100 1100 0011 0000 0111 0100
3	2	0011 0001 0011 0001 1000 0000 0000 1000	19	1	0100 0000 0100 1100 0011 0000 0111 0100
4	1	0100 0000 0100 1100 0011 0000 0111 0100	20	2	0011 0001 0011 0001 1000 0000 0000 1000
5	1	0100 0000 0100 1100 0011 0000 0111 0100	21	2	0011 0001 0011 0001 1000 0000 0000 1000
6	1	0100 0000 0100 1100 0011 0000 0111 0100	22	1	0100 0000 0100 1100 0011 0000 0111 0100
7	5	0000 0000 1000 0000 0100 0001 1000 0010	23	5	0000 0000 1000 0000 0100 0001 1000 0010
8	5	0000 0000 1000 0000 0100 0001 1000 0010	24	2	0011 0001 0011 0001 1000 0000 0000 1000
9	0	1000 0000 0000 0000 0000 0010 0000 0001	25	4	0000 0010 0000 0000 0000 1000 0000 0000
10	3	0000 1100 0000 0000 0000 0100 0000 0000	26	3	0000 1100 0000 0000 0000 0100 0000 0000
11	4	0000 0010 0000 0000 0000 1000 0000 0000	27	3	0000 1100 0000 0000 0000 0100 0000 0000
12	1	0100 0000 0100 1100 0011 0000 0111 0100	28	2	0011 0001 0011 0001 1000 0000 0000 1000
13	1	0100 0000 0100 1100 0011 0000 0111 0100	29	2	0011 0001 0011 0001 1000 0000 0000 1000
14	5	0000 0000 1000 0000 0100 0001 1000 0010	30	1	0100 0000 0100 1100 0011 0000 0111 0100
15	2	0011 0001 0011 0001 1000 0000 0000 1000	31	0	1000 0000 0000 0000 0000 0010 0000 0001

Before an event is processed, its replication-set bitfield is copied into a global variable known as the *CurrentReplicationSet*. This variable represents the current set of replications to be processed by the event. The first replication, called the *FirstReplication*, in the CurrentReplicationSet is also stored in another global variable. It corresponds to the right-most bit that is set to one in the CurrentReplicationSet.

When a replicated state variable is accessed within an event, the bit-field mask stored in the FirstReplication array element is ANDed with the CurrentReplicationSet. This may reduce the collection of replications in the CurrentReplicationSet if the associated replicated values in the model's state have different replication values. When the event processing completes, the CurrentReplicationSet is XORed with the remaining replication set. The result is stored back in the CurrentReplicationSet. If the resulting value is non-zero, then the event is processed again with a new reduced CurrentReplicationSet and FirstReplication. This process continues until all of the bits stored in the CurrentReplicationSet are zero. In this manner, event splitting based on accessing state variables that have different values for different replication sets is automated.

Note that this approach does not require individually examining each replicated state variable as it is accessed to determine which ones have the same value. The bit-field mask automates this very efficiently. However, when an event modifies a replicated value, a check is required to determine if the new value is shared by other replication sets, or if it constitutes a new set. For large numbers of replications, this can be computationally intensive. Performance bottlenecks can occur if the assignment operation is not fully optimized or when multi-replication variable assignments occur frequently. An additional complexity arises when an event modifies state variables and then later accesses other variables causing a reduction in the size of the replication set. Values that were modified but then later eliminated in the event processing cycle must be restored to their original values before reprocessing

the event again with the reduced replication set. This is accomplished in a straightforward manner using the WarpIV Kernel rollback framework to undo those modifications.

Event splitting occurs as an event is processed whenever it accesses state variables that take on different values for the event's replication set. Event splitting causes the event to be processed multiple times for subsets of its original replication set. In the previous illustration, the event used to model Jim ordering his dinner automatically split when he accessed the state variable representing the amount of money he had in his wallet. The HyperWarpSpeed algorithm automatically processed the event twice (once to order chicken, and again to order steak).

Events can potentially be merged to minimize excessive branching and splitting. Merging may occur first before processing events, and then again afterwards for new events that are scheduled. Event merging is an optimization that combines the processing of identical events generated by different branches. A flow chart is provided in Figure 2 showing the HyperWarpSpeed event processing logic.

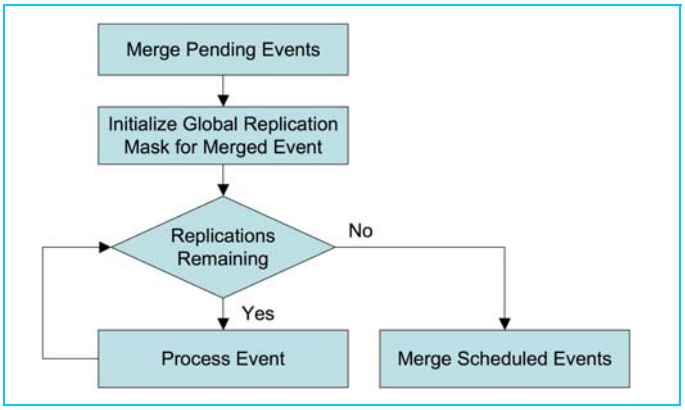

Figure 2: Flow chart showing the event processing sequence in HyperWarpSpeed. An attempt is made to merge events first before processing the current event. The global replication mask is then determined using the collective replication sets of all merged events. The event is then potentially processed multiple times as event splitting occurs. Finally, all generated event messages scheduled by event splitting are potentially merged whenever possible.

HyperWarpSpeed simulations can be configured to receive live inputs that roll back and calibrate the state in real time. If received data does not affect event processing, those events that were rolled back can be rolled forward instead of being reprocessed. Integration of real-time sensor data into Kalman Filters for estimating and predicting enemy behaviors was recently demonstrated to the Air Force. The WarpIV Kernel provides an advanced rollback/rollforward framework that allows events to undo and potentially redo their operations.

3 Parallel Processing

With the multicore revolution underway, tomorrow's software must be written to take advantage of parallel processing resources. Next generation computers may actually run at reduced clock speeds to accommodate large numbers of processors within a single chip. This means that unless software applications are developed to make use of multiple processors, future modeling and simulation programs may actually run slower than they do today. It is therefore critical for next-generation modeling and simulation technologies such as HyperWarpSpeed to fully embrace parallel computing methodologies. The following subsections motivate the need for parallel processing, describe how parallel processing is achieved in the WarpIV Kernel, and discuss open source standards underway for modeling and simulation.

3.1 Next Generation Multicore Computing

Single-processor computing technology has hit the performance wall.[1] Computer experts universally agree that the only way forward in improving run-time performance is through multicore chip designs along with the development of new software applications that are able to take advantage of parallel processing. While Graphics Processing Units (GPUs) have exploited multicore technologies for many years, they are very specialized in their processing capabilities and have limited output bandwidth for gathering the final results.

General-purpose multicore computing capabilities are quickly becoming mainstream. Affordable dual quad-core computers operating at 3 GHz are now available in today's marketplace. More-exotic tiled-core architectures are available.[2] Based on Moore's law, experts predict that processing cores per chip will double every 18 to 24 months. Within a few years, affordable desktop computers will be configured with 16 to 32 processors (or more). By 2021, desktop computers with more than 1,000 cores may be commonplace.

The following excerpt, taken from a recent news article, signifies the importance of developing new scalable software solutions to address the emerging hardware revolution.

> *… Experts predict dire consequences if the software for more complicated applications isn't brought up to speed soon. They warn that programs could suddenly stop getting faster as chips with eight or more cores make their way into PCs. The software as it's currently designed can't take advantage of that level of complexity.*
>
> *"We'd be in uncharted territory," Patterson said. "We need to get some Manhattan Projects going here – somebody could solve this problem, and whoever solves this problem could have this gigantic advantage on everybody else."*
>
> "New Generation of Processors Presents Big Problems, Potential Payoffs for Software Industry."
> Jordan Robertson, Associated Press, July 22, 2007.

Hardware and software experts in the field of parallel computing recognize the significance of the multicore revolution. Yet, there is still much uncertainty concerning the direction that it will actually take. Hardware vendors are actively canvassing the HPC user community for guidance on how applications might best take advantage of next generation chip designs. Balancing processor speeds with shared memory cache architectures presents the greatest challenge. Distributed shared memory with cache coherency is necessary for offering scalability on large multicore systems. To further optimize performance, some hardware vendors have questioned the need for providing cache coherency altogether and would prefer to leave the details of synchronizing shared memory to software developers.[3] However, most software developers are in agreement that simpler parallel programming paradigms are necessary to develop, debug, and maintain multicore programs.

---

[1] Three serious *performance walls* have limited the advancement of single CPU performance. *First*, the power consumed (and heat generated) by increasing the clock speed changes as the square of the clock speed. Current clock speeds are currently at the limit of being practical. *Second*, instruction Level Parallelism (ILP) that exploits vector processing, instruction pipelining, branch prediction, etc. has reached its limit. *Third*, the gap between processing speed and memory access has widened. Because memory access is a serious bottleneck, faster processors will not necessarily produce faster results. Optimal computing performance requires a balanced chip design architecture.

[2] "Tilera Corp. (Santa Clara, Calif.) has said it has started shipping its Tile64 processor, a 64-processor chip based on an architecture that could scale to hundreds and even thousands of cores, according to the company… Production pricing for the Tile64 family starts at $435 in 10,000 unit quantities" EE Times Online.

[3] The IBM Power4 Architecture does not provide cache coherency between processors interacting through distributed shared memory. Furthermore, to improve processing efficiency, instructions might not be processed in the expected order. These issues can lead to situations where one node writes multiple variables in shared memory

The general consensus is that low-level parallel programming is far too challenging for the mainstream software developer. A robust abstract parallel programming framework is necessary to reduce the cost of multicore software development.[4] The WarpIV Kernel provides a parallel and distributed object-oriented computing framework that automates parallel processing on wide variety of hardware and network architectures [19].

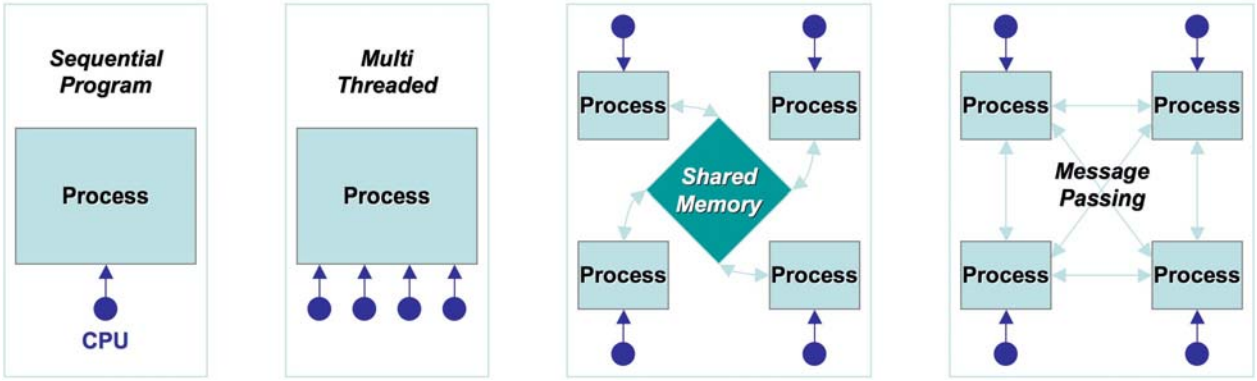

Figure 3: Various types of parallel and distributed computing paradigms.

The supercomputing community has explored various programming approaches over the last twenty years (see Figure 3). These include: (1) vector processing, (2) message-passing, (3) shared memory, (4) threads, (5) grid computing, (6) transactional memory, (7) distributed shared memory and cache coherency, (8) parallelizing compilers, (9) parallel math libraries, (10) distributed object technologies, and (11) parallel discrete-event simulation. Experts uniformly agree that a high-level programming framework is needed to streamline software development. Debugging parallelized software using low-level concurrency primitives such as thread or shared memory locking mechanisms is simply too difficult and expensive for mainstream programs. The WarpIV Kernel provides a high-level programming framework for parallel simulation.

3.2 Optimistic Parallel Discrete Event Simulation

The fundamental challenge of executing a simulation in parallel on multiple processors is to ensure that each object modeled in the simulation processes its events in causally correct ascending time order, while also achieving maximal processing efficiency and concurrency [5]. Optimistic processing techniques such as the Time Warp algorithm [9] are primarily used to support complex parallel simulations that place no constraints on how distributed objects interact. Optimistic simulations allow any object to interact with any other object on any other processing node at any time. This is in contrast to conservative techniques that either (1) limit which objects can interact with which other objects, (2) place constraints on how tightly in time objects on remote nodes can interact with one another, and/or (3) place event ordering rules on inter-object interactions.

Rollback techniques allow objects on any compute node to process their events optimistically assuming that the processing of the next current event will not become invalid due to the arrival of an earlier event scheduled by a simulated object residing on another node. When such straggler messages are received, all optimistically processed events for the receiving object that were processed with time tags greater than the time tag of the straggler event, are rolled back. When using incremental state saving techniques, the events are rolled back in reverse order much like the familiar undo mechanisms that are provided by many commercial software applications.

Events perform two basic operations. They arbitrarily (1) modify state variables and (2) generate new events. Rollbacks must therefore undo those two operations. This means that the rollback infrastructure of the optimistic

---

that are transmitted and then read by other nodes in a different order. So for example, a flag indicating that a variable is safe to read could be handled incorrectly unless the program invokes special memory and/or instruction synchronization services.

[4] Microsoft Corp. sponsored a special *by-invitation-only* workshop on manycore computing on June 20-21, 2007. Participants were the "Who's Who" in the worldwide parallel computing industry. During the workshop, experts uniformly agreed that parallel programming frameworks were needed to simplify the development of parallel applications. Dr. Steinman gave a presentation on the open source standardization efforts that are underway with the WarpIV Kernel.

simulation engine must be capable of (1) undoing the state changes made by each event and (2) retract any new events that were scheduled.

Both incremental and full state saving techniques have been used in the WarpIV Kernel [17] to restore state as rollbacks occur. Full state-saving techniques that are often supported in other systems save the entire state of a modeled object before each event is processed. This can be efficient for objects having small states. However, this approach can become inefficient both in terms of memory consumption and processing overheads for complex objects having large states fragmented over multiple memory segments. The WarpIV Kernel primarily uses automatic incremental state saving techniques that capture each change made to the state as they occur. The rollback infrastructure simply undoes those changes in reverse order. For more complex operations, the rollback infrastructure must not only support the restoration of primitive state variables, but it also must support external I/O, generic dynamic memory allocation/deallocation, and container classes such as trees or lists that store the dynamically allocated memory.

When events are scheduled for objects residing on other nodes, antimessages are sent to retract event-scheduling messages if the event is subsequently rolled back. This can cause cascading rollbacks and further antimessages if those retracted events were processed before they were retracted. Various optimistic flow control techniques have been developed in WarpIV to keep the number of rollbacks and antimessages stable. These techniques include (1) holding back high-risk messages until it is safe (or safer) to release them and (2) limiting event-processing optimism on runaway nodes. A good rollback infrastructure only rolls back those events that are associated with each individual object, and not the entire state of the simulation on a node. Object-based (as opposed to node-based) rollback techniques significantly reduce the number of rollbacks experienced by an application as it executes in parallel.

3.3 Composability Standards for Modeling and Simulation

The WarpIV Kernel provides a sophisticated framework for developing complex parallel and distributed simulations. All of the low-level communications and concurrency mechanisms necessary to achieve scalable performance are built into the high-level event scheduling and processing abstractions. The WarpIV Kernel is a composable top-to-bottom layered architecture that conforms to the Open Modeling and Simulation Architecture (OpenMSA) [18] and Open System Architecture for Modeling and Simulation (OSAMS) [21].

Model composability, in contrast to technology composability, is challenging due to many factors such as conceptual model representations, integrating models with mixed resolutions, and basic integration challenges [4]. The OpenMSA defines a layered architecture of the technologies for parallel and distributed simulation. OSAMS focuses on mixed-resolution model interoperability standards that promote the development of plug and play software model components. Flexible hierarchical composition techniques and abstract interfaces decouple mixed-resolution model components while promoting their full interoperability in a parallel and distributed environment.

A new SISO [15] study group, known as *The Open Source Initiative for Parallel and Distributed Modeling and Simulation* (OSI-PDMS) [16], was formed September 18, 2007 to study such composability architectures with the potential goal of forming new standards. The WarpIV Kernel provides a reference implementation of both the OpenMSA and OSAMS. The HyperWarpSpeed technology and modeling methodology will almost certainly be included in the emerging OpenMSA and OSAMS standards.

4 Real-Time Estimation and Prediction

The conceptual operation of using optimistic simulation to support real-time estimation and prediction is straightforward (see Figure 4). The simulation must always be running and predicting the future up to a specified time or time window, while rolling back affected models occasionally to process real-time inputs. In the WarpIV Kernel, real-time inputs are handled as externally generated events. Inputs can be in the form of *aiding terms* that are used to describe accurately known changes to the system, or in the form of *noisy measurements* that require balancing the believability of the measurements with the system uncertainty specified within the model. In any case, the objects affected by the inputs are rolled back to the current real-world time. Processing these real-time inputs can be thought of as calibrating, or estimating the real-time state of the simulation. After the inputs have been received and processed, the system must quickly rollforward and/or reprocesses all events that were rolled back to continue predicting the future.

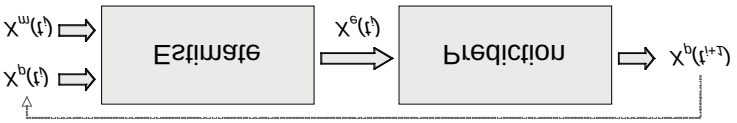

Figure 4: Estimation and Prediction. Noisy measurements are repeatedly combined with previous predictions over time to continually form the best state estimate of the system.

4.1 Kalman Filters

Kalman Filters [23] provide an excellent mechanism to form state estimates and predictions of linear control systems with noisy measurements, accurate inputs, and well understood dynamics. It is assumed that the true dynamical system can be represented as a linear matrix equation of the following form.

$$\overline{X}(t_{i+1}) = \phi(t_{i+1}, t_i)\overline{X}(t_i) + L(t_i) + U(t_i)$$

Where,

- $\overline{X}(t_i)$ is the true state of the system at time t_i.
- $\phi(t_{i+1}, t_i)$ is the transition matrix that extrapolates the state of the system from time t_i to $t_i + 1$.
- $L(t_i)$ is the "aiding" term that describes known inputs to the system between time t_i and $t_i + 1$.
- $U(t_i)$ is the unknown inputs to the system between time t_i and $t_i + 1$.

The true state of the system is never actually known, which is why it is important to perform estimations and predictions based on noisy input measurements and accurately known inputs that are collected regularly over time. Kalman Filters can have different representations depending on whether the estimation and prediction steps are performed separately or collectively in a single step. This section uses the representation of the Kalman Filter that performs the estimation and prediction in a single step. The estimator/predictor form of the Kalman Filter is written as follows:

$$\hat{X}(t_{i+1}) = \phi(t_{i+1}, t_i)\hat{X}(t_i) + L(t_i) + K(t_i)\left[Z(t_i) - H(t_i)\hat{X}(t_i)\right]$$

Where,

- $\hat{X}(t_i)$ is the estimated/predicted state of the system at time t_i.
- $Z(t_i)$ is a vector of measurements of the system taken at time t_i.
- $H(t_i)$ is the measurement matrix that translates the system state to the set of measurements.
- $K(t_i)$ is the Kalman Gain that weighs the correction to the state obtained from each measurement.

The Kalman Gains are computed from the covariance matrix that represents the expectation value of the correlated state errors. The covariance matrix takes into account estimates of the measurement noise and the system-uncertainty noise during each update in time. These equations are shown below.

$$K(t_i) = \phi(t_{i+1}, t_i) P(t_i) H(t_i) \left[R(t_i) + H(t_i) P(t_i) H(t_i)^T \right]^{-1}$$

$$P(t_{i+1}) = \left[\phi(t_{i+1}, t_i) - K(t_i) H(t_i) \right] P(t_i) \phi(t_{i+1}, t_i)^T + Q(t_i)$$

Where, $R(t_i)$ is the measurement noise vector and $Q(t_i)$ is the system noise vector at time t_i. Notice that the gains are small (i.e., approach 0 for simple systems) when $R(t_i)$ gets large compared to $Q(t_i)$. Similarly, the gains become large (i.e., approach 1 for simple systems) when $Q(t_i)$ is much smaller than $R(t_i)$.

Three types of Kalman Filters were developed for this effort: *Range*, *Range-RangeRate*, and *PositionXYZ*. The *Range* filter is useful for predicting the distance between a ground and air target. The *Range-RangeRate* filter extends the *Range* filter by taking into account the derivative of the range, or change in speed, to help the filter converge faster. As opposed to utilizing angle information, the *PositionXYZ* filter efficiently determines the position of a target in space by converting detections to X, Y, Z coordinates.

The Kalman Filters are standalone utilities that provide both rollbackable and non-rollbackable versions, and are defined in the `WpRangeKF.H`, `WpRangeRangeRateKF.H`, and `WpPositionKF.H` header files. The *Range* and *Range-RangeRate* filters inherit from a Kalman Filter base class, defined in the `WpKalmanFilter.H` header file. Useful methods in the Kalman Filter base class provide the ability to (1) update the state of the filter with a measurement and error-sigma at a point in time, (2) extrapolate the filter into the future, (3) print the filter state, and (4) retrieve the time, predicted state, convariance matrix, and Kalman gain. The *PositionXYZ* filter contains three *Range* filters and receives detection updates in the form of a detection class, defined in `WpDetection.H`. The detection class contains truth and perception data, such as the time of detection, sensor position, range, range-rate, direction unit vector, range sigma, range-rate sigma, and direction sigma. Usage of the *Range* Kalman Filter is shown in Code Segment 2, and the corresponding output is shown in Code Segment 3. Additional tests are demonstrated in the `TestKalmanFilter` DeveloperTest.

```
#include "WpRangeKF.H"
#include "WpRandom.H"

int main() {
  WpRangeKF kf(0.1, 0.0); // RangeKF with targetAccelModel = 0.1, alpha = 0.0
  WpMatrix z(1,1);        // Measurement
  WpMatrix zSigma(1,1);   // Error term
  WpRandom random;

  double noise = 0.05;
  double dt = 0.01;
  for (double t=0.0; t<1.0; t+=dt) {
    z(0,0) = sin(t) + random.GenerateGaussian(0, noise);
    zSigma(0,0) = noise;

    kf.UpdateState(dt, z, zSigma);
    cout << "Measurement = " << z(0,0) << endl;
    kf.Print();
  }
}
```

Code Segment 2: Usage of the Kalman Filter. A Range Kalman Filter is utilized to track a noisy data source for one second of time. In this example, the data source is a sine wave containing random Gaussian noise, and is measured every one-hundredth of a second. The error term of the filter is assumed to be constant. Every time step, a measurement is taken and passed to the Kalman Filter. The state of the Kalman Filter is updated and the measurement and predicted state is displayed to standard output.

```
Measurement = 0.434462

X 3 by 1
[0][0] = 0.326902
[1][0] = 2.27894
[2][0] = 6.95614

Measurement = 0.479935

X 3 by 1
[0][0] = 0.36407
[1][0] = 2.43563
[2][0] = 7.24872

Measurement = 0.405163

X 3 by 1
[0][0] = 0.39268
[1][0] = 2.53183
[2][0] = 7.32708

Measurement = 0.444934

X 3 by 1
[0][0] = 0.423448
[1][0] = 2.63555
[2][0] = 7.42621
```

Code Segment 3: Select output of Code Segment 2. Notice how the predicted state, given in X[0][0], lags slightly behind the measured value but improves its accuracy over time. A drastic change in the measured value, caused by a sharp change in the direction of the aircraft, will briefly cause the predicted value to lag behind the measured value.

4.1.1 Matrix Algebra Utility

In order to simplify the mathematics of the Kalman Filter, a utility was developed to perform matrix algebra operations. The WpMatrix class provides matrix addition, subtraction, multiplication, division (i.e. inversion), transposition, and supports operator overloading with both matrices and standard data types. Nested and complex equations, which previously required error-prone loops and temporary variables to solve, are now cleanly represented in a single easy-to-read line of code. To support usage within an optimistic simulation environment, the standalone utility provides both rollbackable and non-rollbackable versions. The rollbackable and nonrollbackable versions are defined in the `RB_WpMatrix.H` and `WpMatrix.H` header files, respectively. The matrix algebra class makes use of the WarpIV ObjectFactory memory management utility, which reuses and recycles memory to avoid unnecessary calls to `new` and `delete`. Basic usage of the matrix algebra class is shown in Code Segment 4.

```
#include “WpMatrix.H”

int main() {
  WpMatrix A(2, 3, "MatrixA");      // Create ‘MatrixA’ [2][3]
  WpMatrix B(3, 2, "MatrixB");      // Create ‘MatrixB’ [3][2]

  A = 0.0;                          // Initialize matrix elements to zero
  A(1,2) = -2.0;                    // Assign a value to element[1][2]

  B = 1.0;
  B(1,1) = -3.5;
  B(2,1) = 1.5;

  WpMatrix C = 1.0 / (A * B + 2.0); // C is the inverse of (A * B + 2.0)
  C.SetName("MatrixC");             // Optionally set the name

  A.Print();                        // Print matrices to standard output
  B.Print();
  C.Print();
}
```

Code Segment 4: Usage of the WpMatrix utility. This example first creates two matrices of different dimensions. The arguments to the matrix constructor are the number of rows, number of columns, and optional matrix name. Operator overloading is used to zero out the matrix in one step, as opposed to iterating through each element. Individual matrix elements are then assigned specific values. Finally, the inverse of the sum of two matrices is stored in a new matrix and the matrices are printed to standard output.

The corresponding output from Code Segment 4 is given in Code Segment 5. Additional capabilities are demonstrated in the `TestMatrix` DeveloperTest, which also verifies correctness of the utility during rollback and rollforward operations.

```
MatrixA 2 by 3
[0][0] = 0
[0][1] = 0
[0][2] = 0
[1][0] = 0
[1][1] = 0
[1][2] = -2

MatrixB 3 by 2
[0][0] = 1
[0][1] = 1
[1][0] = 1
[1][1] = -3.5
[2][0] = 1
[2][1] = 1.5

MatrixC 2 by 2
[0][0] = 0.5
[0][1] = 1
[1][0] = 0
[1][1] = -1
```

Code Segment 5: Output of Code Segment 4. The Print() method displays the matrix name, dimensionality, and element values to standard output.

4.2 Estimation & Prediction Demonstration

Kalman Filters can be used to fuse noisy data from multiple sources. A good example is a tracking system that fuses detections from multiple Radars and IR sensors. This capability has been demonstrated by using real-time detections generated by multiple sensors to estimate aircraft motion within an optimistic simulation environment. This particular demonstration predicted future outcomes based on extrapolating real-time estimates of aircraft states.

The components developed to support this demonstration included (1) a real-world simulation generated a file of detections, and (2) a real-time network-based detection playback system that feeds into (3) an estimation and prediction rollback-based parallel simulation of enemy aircraft in a target-rich theater. Initially, a simulation

consisting of three aircraft, several ground radar, systems and a command center was executed to generate and log potentially noisy aircraft detections to a file.

The real-world simulation animated event diagram is shown in Figure 5. Each aircraft SimObj contains a Component that guides the motion of the aircraft. The aircraft navigate the MCAS Miramar area in San Diego using Great Circle motion, periodically making random changes in their motion and direction in a process loop. The ground radar SimObjs contain a radar Component that periodically scans the area for enemy aircraft in a process loop. A command center SimObj contains a Kalman track fusion Component that processes detections received by the ground radar to form estimates of the position of the aircraft. The command center logs detections to a data file for later playback.

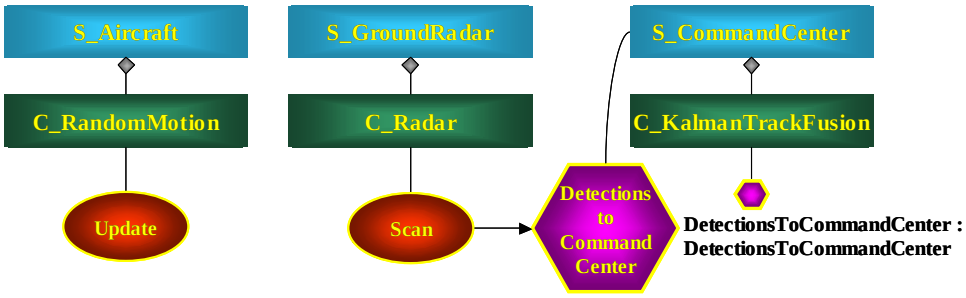

Figure 5: Real-world Simulation Animated Event Diagram.

The estimation and prediction simulation, as shown in animated event diagram in Figure 6, consisted of a detection server and simulation client. The detection server played back and sent the logged detections in real-time to the simulation client. The simulation continually processed the detections with a Kalman filter component to form estimates of the position of the aircraft in real time. In order to generate rollbacks, additional events were scheduled by the aircraft to bomb static targets.

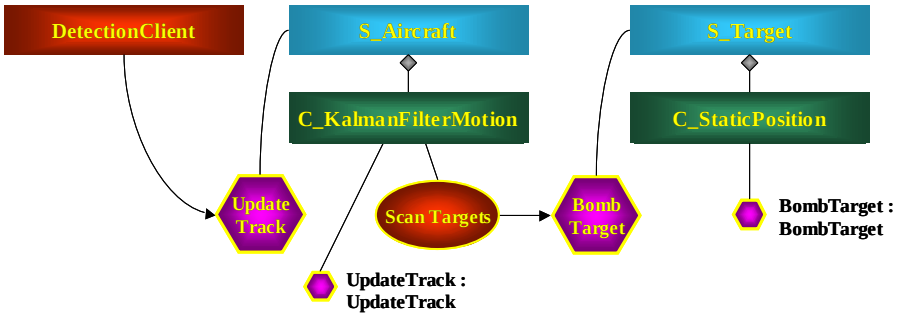

Figure 6: Estimation/Prediction Simulation Animated Event Diagram.

As an alternate view, the sequence diagram for the estimation and prediction simulation is shown in Figure 7.

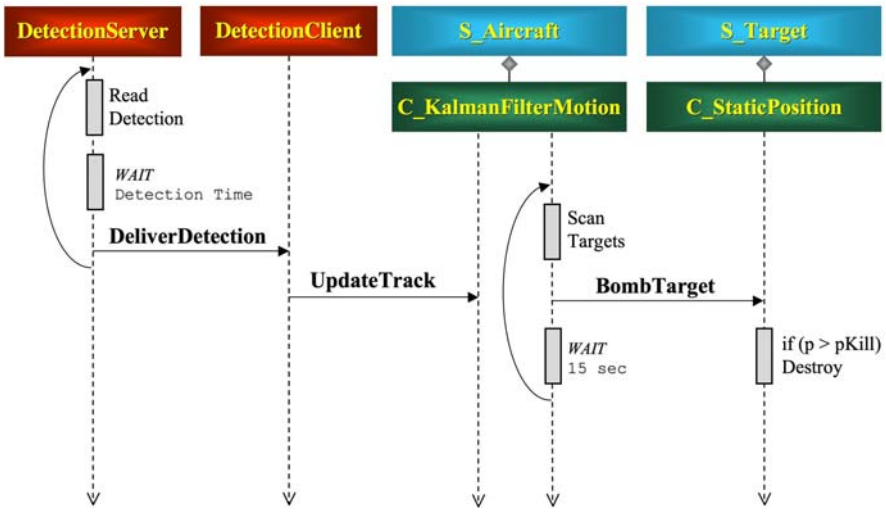

Figure 7: Estimation/Prediction Simulation Sequence Diagram.

The Kalman filter error was saved to disk for later analysis. Figure 8 plots the measurement error (blue) and estimate error (green) as a function of time for an enemy aircraft as captured during the real-world simulation. Spikes in the estimate error denote when the aircraft made random changes to its motion.

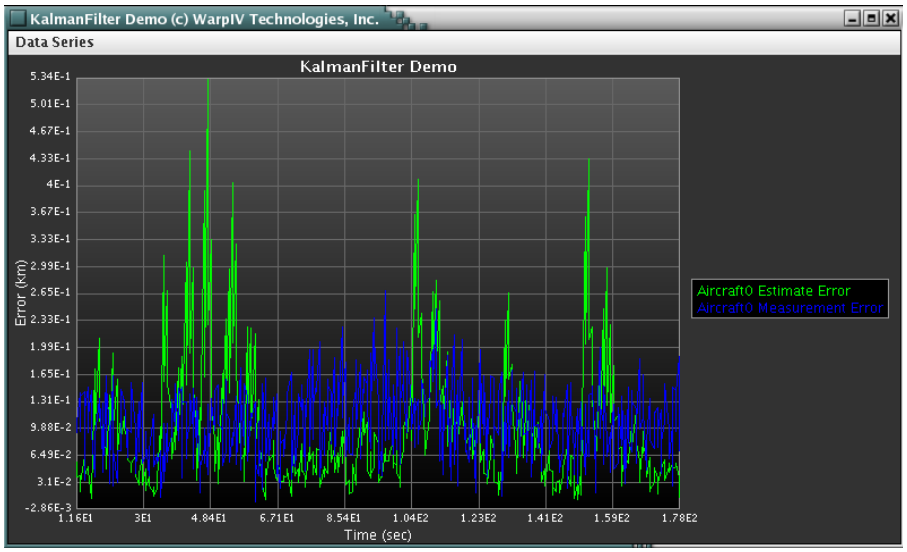

Figure 8: Real-world Simulation Kalman Filter error plot for an enemy aircraft. Measurement error is shown in blue, and estimate error is shown in green for the duration of the simulation. Spikes in the estimate error denote points in time in which the aircraft abruptly changed its motion.

Calibrating the simulation with live data ensures that reasonable predictions are generated and eliminates the need to void a potentially invalid execution. Predicting enemy intents and behaviors based on past and current data [14] is a much more challenging topic that is well beyond the scope of this report [1].

5 Statistical Algebra

A critical component of the real-time estimation and prediction strategy described in this report is to use multiple hypotheses branching to split the event processing into a series of replications that are internally managed within the parallel simulation execution. If random numbers are directly used in the models, then the execution will quickly splinter into large numbers of replications because the state variables rapidly diverge. Therefore, it is important to only branch when critical decisions that have very different state representations are required. To handle statistical models, the statistical algebra capabilities described here can be used to represent statistical state variables as distributions, not just random values.

Convolution operations are required to obtain new distributions from algebraic expressions involving two independent distributions. If *A(x)* and *B(y)* each represent distributions over independent variables *x* and *y*, then *P(z)* for a function *F(A,B)* can be represented as follows:

$$P(z) = \int_{x=-\infty}^{x=+\infty} A(x)dx \int_{y=-\infty}^{y=+\infty} B(y)\delta(z - F(x,y))dy$$
$$= \int_{x=-\infty}^{x=+\infty} A(x)B(F^{-1}(x,z))dx$$

Where $F^{-1}(x,z)$ is the inverse of $F(x,y)$ that is determined by solving $z=F(x,y)$ for y. Note that $\delta(z-F(x,y))$ represents the *Dirac Delta Function* whose value is zero when its argument is non-zero, is infinite when its argument is zero, and is normalize to have the area under its curve equal to one.

For the simple case where $F(A,B) = A + B$, the inverse is found by solving $z = x + F^{-1}(x,z)$. The inverse is simply $z - x$, which is the familiar convolution shown in the equation below.

$$P(z) = \int_{-\infty}^{+\infty} A(x)B(z - x)dx$$

The current WarpIV Statistical Algebra implementation represents distributions through the use of discrete bins ranging over their independent variable. Applications may define the number of bins and the domain of the independent variable. The sum of the binned values is normalized to one. Numerically computing algebraic expressions involving distributions is accomplished by summing the weighted contribution to the appropriate new bin. Special care over binning and sub-binning must be taken into account to ensure that the resulting distributions obtained from algebraic operations are smooth. Otherwise, potential spikes and gaps can arise in the resulting distributions.

Using operator overloading in C++, the WarpIV Statistical Algebra capability automatically supports basic algebraic operations such as `+`, `-`, `*`, and `/`. However, for further generality, it also supports transcendental functions such as `sqrt()`, `sin()`, `cos()`, `tan()`, `arcsin()`, `arcos()`, `arctan()`, `exp()`, `log()`, `pow()`, etc. Special care has been taken to ensure that the range of independent variables does not contain any singularities or undefined values. For example, if the independent variable *x* for distribution *A* ranges from -1.0 to +1.0, then singularities would arise in the expression *B=1/A*. In a similar manner, square roots of negative numbers are currently not allowed. Support for complex variables may be provided in the future.

Handling correlations in more complex equations is also important. For example, there is a difference between the expressions *Y = pow(X, 2)* verses *Y = X*X*. In the first expression involving the power transcendental function, the argument *X* is correlated. However, the second algebraic expression multiplies *X* by itself in a manner that ignores correlations. Thus, these two algebraic expressions yield very different resulting distributions. Further automating correct handling of correlations were explored in this effort.

Additional capabilities are provided by the WarpIV Statistical Algebra package to allow applications to smooth and/or re-bin distributions, fit standard statistical functions to numerical distributions, obtain statistical metrics such as the mean and standard deviation from distributions, built-in support for a wide variety of common statistical functions, and perform correlated operations on user-defined functions involving multiple distributions.

While the Statistical Algebra package in WarpIV was originally developed to support the ideas in this report, it is important to point out that this capability can be used in a wide variety of other applications. The Statistical

Algebra package can be used to explore the statistical properties of basic models such as standalone Kalman Filters used on airborne radar tracking systems.

An example showing the use of the WarpIV Statistical Algebra package is shown in the figures below. In this example, it is assumed that *X* is a Gaussian distribution, *Y* is an exponential distribution, and *Z* is computed using the following equation.

$$Z = \frac{1}{\pi}\log\left(\sqrt{X} + 2\cos(Y) + 5\right)$$

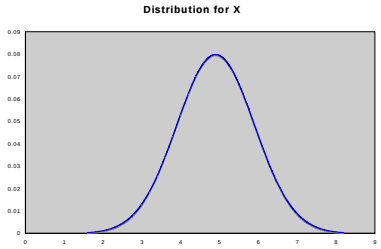

Figure 9: Gaussian distribution for variable X.

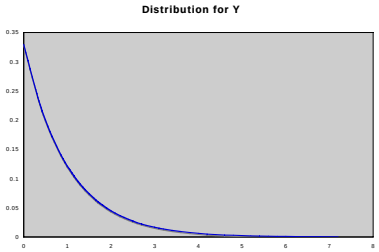

Figure 10: Exponential distribution for the variable Y.

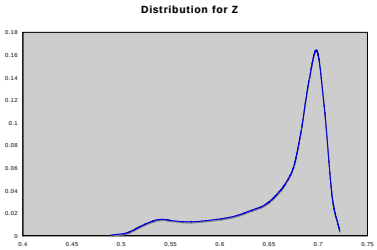

Figure 11: Computed distribution for the variable Z.

The Statistical Algebra library was extended in this effort to support mathematical functions and operations on up to five correlated distributions. The `CorrelatedFunction()` is defined in the `WpStatDouble.H` header file. The function accepts up to five WpStatDouble distributions, and a pointer to the function performing the mathematical operation on the distributions. A new WpStatDouble distribution is returned as a result. Operations on non-correlated distribution perform a convolution, whereas operations on correlated distributions do not. For example, consider squaring a distribution. The non-correlated approach would multiply bin_i of the distribution by itself and all other bin indices to determine the probability for the bin, requiring n^2 computations. The correlated approach only multiplies bin_i by itself to determine the probability, requiring n computations.

The following example demonstrates both approaches in Code Segment 6. This example creates a Gaussian distribution with a mean of 0.5, standard deviation of 0.1, and 50 bins. A square of the distribution is created using both non-correlation and correlation methods. The first result squares the distribution via operator overloading, which internally assumes the distributions are independent and thus not correlated. The second result squares the distribution using the correlation function. The output from Code Segment 6 is shown in Code Segment 7.

```
#include "WpStatDouble.H"
#include "WpStatGaussian.H"

double Square(double x) { return x * x; }

int main() {
  WpStatGaussian g(0.0, 1.0, 50, 0.5, 0.1, "Gaussian");
  g.GraphDistribution();

  WpStatDouble gg = g * g; // Operator overloading assumes non-correlation
  gg.SetName("x * x");
  gg.GraphDistribution();

  WpStatDouble gSquared = CorrelatedFunction(Square, g); // Correlated approach
  gSquared.SetName("Square");
  gSquared.GraphDistribution();

  return 0;
}
```

Code Segment 6: Statistical Algebra correlations. A Gaussian distribution is squared to demonstrate usage of both non-correlated and correlated approaches. Operator overloading assumes non-correlation. Squaring the Gaussian distribution via correlation requires a call to the CorrelatedFunction(). The CorrelatedFunction() requires a pointer to the function performing the mathematical operation on the Statistical Algebra distribution, and the Statistical Algebra distribution itself. A new WpStatDouble distribution is returned as a result.

```
Distribution of Gaussian (InstanceId = 1)
Mean = 0.5, Standard deviation = 0.0995859

       0.16 | (0.000344741)
       0.18 |- (0.000653795)
        0.2 |- (0.00119129)
       0.22 |-- (0.00208556)
       0.24 |---- (0.00350798)
       0.26 |------ (0.00566915)
       0.28 |--------- (0.00880253)
        0.3 |------------ (0.0131318)
       0.32 |------------------ (0.0188222)
       0.34 |------------------------- (0.0259206)
       0.36 |--------------------------------- (0.0342964)
       0.38 |----------------------------------------- (0.0435992)
        0.4 |---------------------------------------------------- (0.0532522)
       0.42 |------------------------------------------------------------ (0.062492)
       0.44 |------------------------------------------------------------------- (0.0704596)
       0.46 |---------------------------------------------------------------------------- (0.0763279)
       0.48 |---------------------------------------------------------------------------- (0.0794429)
        0.5 |---------------------------------------------------------------------------- (0.0794429)
       0.52 |---------------------------------------------------------------------------- (0.0763279)
       0.54 |------------------------------------------------------------------- (0.0704596)
       0.56 |------------------------------------------------------------ (0.062492)
       0.58 |---------------------------------------------------- (0.0532522)
        0.6 |----------------------------------------- (0.0435992)
       0.62 |--------------------------------- (0.0342964)
       0.64 |------------------------- (0.0259206)
       0.66 |------------------ (0.0188222)
       0.68 |------------ (0.0131318)
        0.7 |--------- (0.00880253)
       0.72 |------ (0.00566915)
       0.74 |---- (0.00350798)
       0.76 |-- (0.00208556)
       0.78 |- (0.00119129)
        0.8 |- (0.000653795)
       0.82 | (0.000344741)

Distribution of x * x (InstanceId = 2)
Mean = 0.250079, Standard deviation = 0.0713325

      0.0656 |- (0.00201012)
      0.0856 |----- (0.00686199)
      0.1056 |----------- (0.016604)
      0.1256 |---------------------- (0.0322163)
      0.1456 |----------------------------------- (0.0528355)
      0.1656 |------------------------------------------------- (0.0753512)
      0.1856 |------------------------------------------------------------- (0.0952851)
      0.2056 |---------------------------------------------------------------------------- (0.108462)
      0.2256 |---------------------------------------------------------------------------- (0.112892)
      0.2456 |---------------------------------------------------------------------------- (0.108474)
      0.2656 |------------------------------------------------------------- (0.0970541)
      0.2856 |------------------------------------------------------ (0.081443)
      0.3056 |---------------------------------------------- (0.0646931)
      0.3256 |-------------------------------- (0.0485236)
      0.3456 |---------------------- (0.0345337)
      0.3656 |---------------- (0.0238466)
      0.3856 |---------- (0.0155879)
      0.4056 |------- (0.00981892)
      0.4256 |---- (0.00602318)
      0.4456 |-- (0.00350051)
      0.4656 |- (0.00200923)
      0.4856 |- (0.00109737)
      0.5056 | (0.000575157)
      0.5256 | (0.000301411)

Distribution of Square (InstanceId = 3)
Mean = 0.259822, Standard deviation = 0.100462

      0.0256 |-- (0.00171248)
   0.0450286 |----- (0.00497145)
   0.0644571 |---------- (0.0109394)
   0.0838857 |------------------ (0.019084)
    0.103314 |------------------------------ (0.0307575)
    0.122743 |---------------------------------------- (0.0418471)
    0.142171 |------------------------------------------------- (0.0539836)
      0.1616 |---------------------------------------------------------- (0.0649034)
    0.181029 |------------------------------------------------------------------- (0.0725503)
    0.200457 |---------------------------------------------------------------------------- (0.0772933)
    0.219886 |---------------------------------------------------------------------------- (0.0788556)
    0.239314 |---------------------------------------------------------------------------- (0.077394)
    0.258743 |------------------------------------------------------------------- (0.0734097)
    0.278171 |---------------------------------------------------------- (0.0675952)
      0.2976 |------------------------------------------------------ (0.0606685)
    0.317029 |------------------------------------------------- (0.0532481)
    0.336457 |---------------------------------------- (0.0438597)
    0.355886 |------------------------------- (0.0366649)
    0.375314 |--------------------------- (0.030325)
    0.394743 |----------------------- (0.0247889)
    0.414171 |------------------- (0.0193782)
      0.4336 |------------- (0.0142409)
    0.453029 |---------- (0.0113721)
    0.472457 |--------- (0.00893531)
    0.491886 |------ (0.00602462)
    0.511314 |----- (0.00459467)
    0.530743 |---- (0.00357512)
    0.550171 |-- (0.00227287)
      0.5696 |-- (0.00171011)
    0.589029 |- (0.00131446)
    0.608457 |- (0.000732777)
    0.627886 |- (0.00060457)
    0.647314 | (0.000392227)
```

Code Segment 7: Output of Code Segment 6. The first distribution represents the original distribution, which was automatically re-binned to provide a better representation. Notice the slight difference in the shape, mean, and standard deviation between the non-correlated (second) and correlated (third) distributions. While similar, the distributions are not identical.

5.1 Statistical Analysis of Simulation Results

An important component of the HyperWarpSpeed approach is to perform statistical analysis of intermediate or final results. This is accomplished by examining the distribution of various measures of effectiveness for the set of replications of interest (i.e., the ones that resulted from going down specific branches). The WarpIV Kernel S*tatistical Algebra* package [22] supports algebraic operations (e.g., convolutions, etc.) using operator-overloading techniques and transcendental functions on variables representing statistical distributions. An interface has been developed to convert multi-replication variables into statistical algebra variables. The interface allows statistical algebra distributions to be formed using the multi-replication variable and a replication mask that only includes values specified by the mask. Then, statistical analysis to determine the effectiveness of various plans can be performed. The results may then feed back into plan optimization, where plans that are not effective can be pruned. This can provide a learning environment where better decisions are made over time as various branches are explored and outcomes are determined.

Basic usage of this capability is demonstrated in Code Segment 8 and the `TestMultiVarHist` DeveloperTest. Usage requires initializing a WpStatDouble object with the multi-replication variable, number of histogram bins, and optional replication mask.

```
#include "WpStatDouble.H"
#include "RB_WpMultiVar.H"

int main() {
  int multiIntValues[WP_MULTI_VAR_MAX_REPLICATIONS];
  for (unsigned i=0; i<WP_MULTI_VAR_MAX_REPLICATIONS; i++) {
    multiIntValues[i] = i / 4;
  }

  int mask[WP_MULTI_VAR_MASK_ARRAY_SIZE];
  for (unsigned i=0; i<WP_MULTI_VAR_MASK_ARRAY_SIZE; i++) {
    mask[i] = 0;
  }

  WpSetMultiMaskRep(18, mask);         // Set specific replications in the mask
  WpSetMultiMaskRep(49, mask);
  WpSetMultiMaskRep(59, mask);

  RB_WpMultiInt multiInt;
  multiInt.Init(multiIntValues);       // Initialize the multi-int w/ an array of values

  WpStatDouble statDouble;
  statDouble.Init(multiInt, 10);       // Pass in the multi-int variable and # of bins
  statDouble.Print();
  statDouble.Init(multiInt, 10, mask); // Pass in the multi-int, # of bins, and mask
  statDouble.Print();
}
```

Code Segment 8: Forming Distributions of Multi-Replication Variables. First, a multi-replication integer is initialized with an array of values, and specific replications of interest are set in a multi-replication mask. Then, a WpStatDouble instance is created, and initialized with the multi-replication integer, number of bins, and optional mask. Finally, a print method displays the results to standard output.

Output from Code Segment 8 is shown in Code Segment 9. The first half of the output is a histogram representing the distribution of values for each replication in the multi-integer data type. The output specifies the midpoint, probability density, and cumulative density for each histogram bin. This example specified 10 bins. The second half of the output yields a histogram representing the distribution of values for the replications specified in the mask.

```
WpStatDouble (0, 0.125), F = 0.125
WpStatDouble (3.2, 0.09375), F = 0.21875
WpStatDouble (6.4, 0.09375), F = 0.3125
WpStatDouble (9.6, 0.09375), F = 0.40625
WpStatDouble (12.8, 0.09375), F = 0.5
WpStatDouble (16, 0.125), F = 0.625
WpStatDouble (19.2, 0.09375), F = 0.71875
WpStatDouble (22.4, 0.09375), F = 0.8125
WpStatDouble (25.6, 0.09375), F = 0.90625
WpStatDouble (28.8, 0.09375), F = 1

WpStatDouble (4, 0.333333), F = 0.333333
WpStatDouble (5.1, 0), F = 0.333333
WpStatDouble (6.2, 0), F = 0.333333
WpStatDouble (7.3, 0), F = 0.333333
WpStatDouble (8.4, 0), F = 0.333333
WpStatDouble (9.5, 0), F = 0.333333
WpStatDouble (10.6, 0), F = 0.333333
WpStatDouble (11.7, 0.333333), F = 0.666667
WpStatDouble (12.8, 0), F = 0.666667
WpStatDouble (13.9, 0.333333), F = 1
```

Code Segment 9: Output from Code Segment 8.

6 Measures of Effectiveness and Performance

The primary purpose of the estimation and prediction techniques discussed in this report is to accurately predict the future based on simulation models that are calibrated using real-time inputs to estimate the current state of the system. It is also important to define metrics that characterize the performance and effectiveness of the overall system and to provide warnings when unacceptable circumstances arise. These metrics provide the feedback to the user indicating the quality of the outcome based on real-time estimation and prediction. They are critical in supporting system optimization techniques. In this section, all metrics are normalized between zero and one. A score of one would indicate perfect operation, while a score of zero would indicate complete failure of the system or planned outcome.

It is important to distinguish the difference between the terms *effectiveness* and *performance*. Usually the term effectiveness relates to the overall results of the simulation and is used to indicate how well the objectives were met. The term performance usually relates to how well systems or subsystems performed their function. For example, an aircraft might perform poorly while still effectively meeting its mission objectives. It is possible for the simulated system to not perform according to plan, while still producing an effective result. A good example of this in sports is a broken play in football that still results in scoring a touchdown. In this case, the play was poorly executed (low score for the performance) while the end result still produced a touchdown (high score for the effectiveness).

Battle plans in a complex military operation evolve over time. Anticipated gains and losses therefore evolve. It is important to not just look at the raw effectiveness metric at any point in time to characterize the operation of the plan. For example, the invasion at Normandy on D-Day started out very poor in terms of losses to the allied forces, yet those losses were anticipated. Of course the plan in the long run was very effective and concluded with a complete victory and the liberation of Europe. Like a chess game, it is sometimes advantageous to sacrificing a pawn early on to obtain a better position that later wins the battle. This leads to defining two terms: (1) raw effectiveness and (2) relative effectiveness. These terms will be described in the context of a battlefield simulation involving red and blue opposing forces.

6.1 Raw Effectiveness

Raw effectiveness is determined by assigning an intrinsic value to each blue or red asset in the battle. The normalized raw effectiveness score is determined by summing these values in a meaningful way. A score of one would indicate that all red assets have been destroyed without any blue losses. Similarly, a score of zero would indicate that all blue assets have been destroyed without any red losses.

The intrinsic value for each entity (or asset) in the battle can be defined at a given point in time. Normally, the intrinsic value does not change for each entity. However, it is possible for new entities to enter the battle, for damaged entities to be repaired, and/or for new information to be provided indicating changes to the intrinsic value of an entity. Intrinsic values can be defined as follows:

$I_i^{[B,R]}(t)$ = Intrinsic value for [blue,red] $entity_i$ at time t

The actual value for each entity (or asset) in the battle can be defined at a given point in time. The actual value ranges from zero (meaning that the entity/asset has been destroyed) to the full intrinsic value (meaning that the entity/asset has perfect health, has not diminished its capacity to engage in battle, has all of its weapon systems in tact, and has a full fuel supply).

$A_i^{[B,R]}(t)$ = Actual value for [blue,red] $entity_i$ at time t

The total intrinsic and actual values for blue and red entities/assets in the battle can be specified at a given point in time:

$$I^{[B,R]}(t) = \sum_i^{N_{[B,R]}} I_i^{[B,R]}(t)$$

$$A^{[B,R]}(t) = \sum_i^{N_{[B,R]}} A_i^{[B,R]}(t)$$

As previously mentioned, raw effectiveness is a value between zero and one indicating the raw effectiveness of the plan outcome. A value of one indicates complete success (i.e., all red entities/assets destroyed and no blue entities/assets destroyed). A value of zero indicates complete failure (i.e., all blue entities/assets destroyed and no red entities/assets destroyed). The raw effectiveness is not an indication of how accurately the plan has been followed. It is simply a measure of the overall outcome.

$$RawEffectiveness(t) = \frac{A^B(t) + \left[I^R(t) - A^R(t)\right]}{I^B(t) + I^R(t)}$$

A successful plan normally results in the raw effectiveness value generally increasing over time. However, it is possible for a plan to accomplish its mission, even though the final raw effectiveness decreases. This would happen if the cost of completing a mission turns out to be higher than the overall gain.

Because of the uncertain nature of predicting the outcome of plans, it is important to execute multiple replications or support multiple-hypothesis branch points and statistically analyze the results of the different replications. The mean and standard deviation of these replications may be provided at regular time increments during the simulation. These mean values and their standard deviations can be fitted using χ^2 analyses to obtain time-based visual plots that provide trend analysis visualization to users.

The raw effectiveness can be thought of as the overall measure of effectiveness of the plan. This is different from the relative effectiveness, which is a measure of the plan performance (i.e., how accurately the plan was followed).

6.2 Relative Effectiveness

The state of each entity/asset in the plan at any point in time can be defined as an abstract vector of values. These state values can be projected forward in time either from simulation or from the actual planned expectations. In air combat operations, this can come directly from the Air Tasking Order (ATO). As discussed earlier in this technical section, estimated state values are provided through the processing of live data feeds that are provided as inputs into the system. These three kinds of state vectors are defined below.

$\vec{X}_i^{[B,R]}(t)$ = Predicted state vector of [blue,red] $entity_i$ at time t

$\vec{Y}_i^{[B,R]}(t)$ = Projected state vector of [blue,red] $entity_i$ at time t

$\vec{Z}_i^{[B,R]}(t)$ = Estimated state vector of [blue,red] $entity_i$ at current time t

The relative effectiveness is a measure of how accurately the plan is being performed with respect to the expectations or of the plan. In other words, the relative effectiveness compares *X* with *Y*. Two types of relative effectiveness are computed: (1) simulated predictions in time vs. planed expectations from the plan, and (2) estimation of the system state at the current time with respect to the real-time picture.

In the first case, the relative effectiveness provides a prediction of the uncertainty of the plan performance over time. It predicts when the plan might fall apart, and when new planning may be required. It provides insight on the chaos that may ensue during the fog of war. Some replications may deviate from the plan due to statistical and/or uncertainties in the scenario, while other replications produce the anticipated outcome. A statistical analysis of the relative effectiveness is used to help characterize the plan dispersion. A mean and sigma for the relative effectiveness are computed for each regular time increment in the simulation. Like the raw effectiveness, the mean and standard deviations for the relative effectiveness can also support χ^2 curve fits and trend analysis plots.

In the second case, the relative effectiveness allows simulation replications that began their execution in the past to verify that they match the current real-time picture. The relative effectiveness for this second case is used to prune those replications or branches that do not match the real world picture. These two cases are shown below:

Case 1 – Simulated predictions:

$$RelEffectiveness(t) = 1 - \left(\frac{1}{I^B + I^R}\right)\sum_i^N \left|\vec{X}_i(t) - \vec{Y}_i(t)\right|$$

Case 2 – Simulated estimates:

$$RelEffectiveness(t) = 1 - \left(\frac{1}{I^B + I^R} \right) \sum_i^N \left| \vec{X}_i(t) - \vec{Z}_i(t) \right|$$

Note that for each of these cases, the magnitude of the vector difference is weighted for each vector value. The overall magnitude for each term in the sum is normalized and lies between zero and its intrinsic value. Computing the normalized state vector difference between the expected state and the predicted or estimated state is beyond the scope of this discussion. However, characterizing how much a plan has deviated will likely involve issues such as: (1) location of the entity with respect to where it is supposed to be, (2) deviation of the plan and allowing for time considerations such as being behind or ahead of schedule, and (3) health of the entity with respect to the planned health. Characterizing the state differences is an active topic of research and development.

However, a score of one would indicate that the plan is being executed perfectly. It is important to remember that losses to assets may be expected, so a score of one does not imply the best possible outcome. A score of zero would indicate complete chaos, meaning that the plan has fallen apart and is no longer valid. Like a broken football play, it is still possible to achieve the objectives even for a low relative effectiveness score. For Case 2, a low score would indicate that the replication or branch does not agree with reality and should therefore be pruned.

6.3 Supporting Software Framework

SimObj and Component classes within the WarpIV Kernel were modified to include rollbackable state variables identifying their force type, health value, and intrinsic value. Possible force types currently include blue, red, white, and unknown, and are enumerated as `WP_FORCE_TYPE_<BLUE, RED, WHITE, UNKNOWN>`. Entities representing white and unknown force types are excluded from MOE/MOP computations. If a SimObj contains one or more Components, the Components are assumed to be the same force type as the SimObj. The health value is a double ranging from `0.0` to `1.0`, and the intrinsic value is a double ranging from `0.0` to `MAX_DOUBLE`. The intrinsic value determines how much weight the health of a SimObj carries on the raw effectiveness calculation.

The force type, intrinsic value, and health can be set during initialization and modified during run-time. `GetHealth()` and `GetIntrinsicValue()` methods are provided to query the current health and intrinsic value of SimObjs and Components, respectively. A `GetForceType()` method is provided to query the force type of a SimObj. Likewise, `SetHealth()`, `SetIntrinsicValue()`, and `SetForceType()` methods are provided to assign or modify the value of these attributes. Code Segment 10 demonstrates initializing these attributes for an Aircraft SimObj in a run time class input configuration file. Note that for simplicity, the run time class permits force types to be defined as `Blue`, `Red`, `White`, and `Unknown`.

```
class SimObjs {
  reference AIRCRAFT Aircraft1
}

class Aircraft1 {
  string ForceType Red
  double IntrinsicValue 5.0
  double Health 1.0
}
```

Code Segment 10: Initializing the Force Type, Intrinsic Value, and Health of a SimObj in a RTC input file.

The raw effectiveness equation requires the actual intrinsic value for each SimObj. A SimObj's actual intrinsic value is simply the product of its intrinsic value and health. If the SimObj contains one or more Components, the actual intrinsic value of the SimObj is the sum of the actual value of its Components. Actual intrinsic values are automatically computed as the health of the SimObj or one of its Components is modified. Computing the actual intrinsic value for each SimObj is trivial; however, computing the raw effectiveness in real-time for a distributed rollbackable simulation is not so easy.

A framework was implemented to automatically compute measures of effectiveness and measures of performance within a WarpIV simulation during execution. To do so required synchronizing all nodes to ensure the calculation takes place from the same point in time. A Global MOE utility, defined in the `WpGlobalMoe.H` header file, provides a simple interface for computing the global raw effectiveness. Implementing this feature in a WarpIV

simulation requires registering a user function in the main processing loop to carry out the computation. The user function is internally called after every event is processed (a future optimization could limit how frequently the user function is called). The `TestTracking` DeveloperTest demonstrates how to compute the global raw effectiveness.

Analyzing the raw effectiveness of a plan after simulation completion required a different approach. This approach is detailed in Section 6.4.

6.4 Data Logging Utility

Supporting after action analysis of a plan requires saving the state of simulation variables to disk. One may think the simplest approach to recording the raw effectiveness would be logging the value in a time-step manner. However, recall that computing the global raw effectiveness requires costly and inefficient synchronization with all processing nodes used in the simulation. A more efficient approach would avoid synchronization altogether and simply record the actual intrinsic values for each SimObj and Component to the trace file whenever they change. Then, these values could be parsed from the trace file to compute the global raw effectiveness. This event-driven approach reduces the amount of data saved to disk and allows the simulation to execute as-fast-as-possible without interruption.

Data logging capabilities were developed in the WarpIV Kernel to enable users to log specific data to a trace file during simulation execution. Support is provided for single values and arrays of Boolean, byte, double, integer, and string data types. Logging data to a trace file from an event requires a single call to a macro. Macros for defining data types and arrays are of the form `LOG_<DATA_TYPE>` and `LOG_<DATA_TYPE>_ARRAY`, respectively. Data logging capabilities are defined in the `WpLogData.H` header file. After the macro is called, each value or array is internally captured as a rollbackable data type. The data is written to disk after the GVT of the simulation exceeds the time of the event, thus eliminating the possibility of a rollback that would save duplicate or redundant data to the file. Code Segment 11 demonstrates modifying the health of a SimObj and logging an integer array to disk.

```
#include “C_Pilot.H” // A user-defined Component
#include “WpLogData.H”

void C_Pilot::AircraftBombed() {
  double health = GetHealth();
  SetHealth(health - RANDOM.GenerateDouble(0.0, health));

  int x[4] = {0, 1, 2, 3};
  LOG_INT_ARRAY("MyIntArray", x, 4) // Log the integer array

  SCHEDULE_ThresholdScan(SIM_TIME + 10.0, this);
}
```

Code Segment 11: Data Logging Usage. An array of integers is created and initialized inside of an event method. The data is easily logged to the trace file by a single call to a macro. Logging an integer array requires passing a (1) character string description of the data, (2) pointer to the array, and (3) array size. The description is used for identifying and retrieving the array from the trace file.

Trace files record data from events in accordance with the WarpIV Run Time Class (RTC) format. The RTC format is easy to read, interpret, and parse with the WarpIV Class Reader utility. Trace data from each event is logged in its own class. Nested classes can exist inside of the main class to help organize different types of data. The specific RTC record of the event generated from Code Segment 11 is given in Code Segment 12.

```
class <50:0:0:24:0>AIRCRAFT[0:0:1]::AircraftBombed {
  double CPU 4.41084e-05

  reference LOCAL_EVENT <60:0:0:24:0>AIRCRAFT[0:0:1]::ThresholdScan

  class LoggedData {
    double IntrinsicValue 5
    double ActualIntrinsicValue 0.0172804
    int MyIntArray {
      0
      1
      2
      3
    }
  }
}
```

Code Segment 12: Sample RTC Record in a Trace File. The "MyIntArray" integer array is recorded as a member of the LoggedData class in the trace file.

This particular record logged data during an `AircraftBombed` event. The class name of the record indicates the (1) time of the event, (2) type and object handle of the simulation object executing the event, and (3) name of the method implementing the event. Records inside the class automatically include timing information, a re-entry label if the event contains a process, and references to any subsequent events scheduled by the event. The logged data section recorded the integer array and automatically captured the intrinsic and actual intrinsic health values since the health was modified in the event. To provide a visual representation of the raw effectiveness and logged data, a trace file analyzer was developed. The trace file analyzer is discussed in Section 6.5.

6.5 Trace File Analysis

A Java-based plotting utility, shown in Figure 12, was previously developed for the WarpIV Kernel to support after action review and real-time plotting capabilities. The platform-independent utility permits the generation of any combination of 2D scatter plots, lines, area plots, and histograms. The interactive plot enables users to hover over data points to display data values, zoom in and out, optionally set the data capacity for real-time plots, adjust data transparency settings, and move a data series to the front or back of the canvas to better visualize overlapping histogram and area plots. The plotting canvas was developed as a standalone Java Component (JComponent) that can easily be imbedded in any Java-based application. Several APIs are provided to control the appearance and behavior of the plotter.

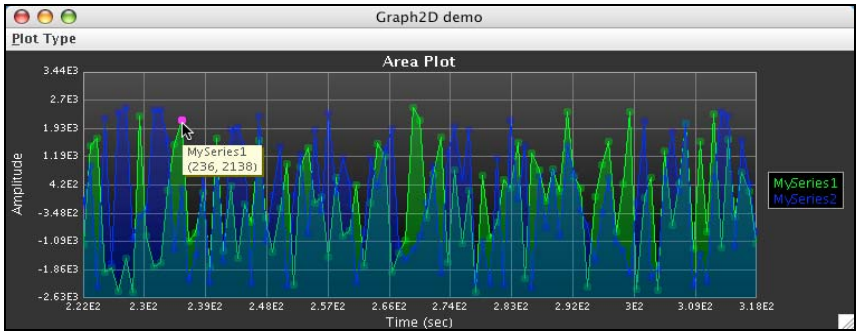

Figure 12: Java-based 2D Plotter Component. This example graphs an area plot, in real-time, of two independent streams of data. Transparency settings were modified to better visualize the data series in the background. Adjusting transparency is important for viewing stacked histograms or distributions of data. As an alternative, the ordering of the data series could be adjusted to bring data of interest from the background to the foreground.

The plotting utility was leveraged to develop a trace file analysis tool, shown in Figure 13. Given a trace file name, the tool automatically parses the file to compute the raw effectiveness each time the actual intrinsic value for a simulation object changes. The actual intrinsic values of each simulation object are automatically plotted as a function of time, along with the raw effectiveness. The user has the ability to select which simulation objects to display in the plot. *Note that another WarpIV Technologies effort has further extended this capability to plot*

anything logged during a simulation. Logged data is organized in a selectable hierarchical tree structure by simulation object type, enumeration Id, and attribute type.

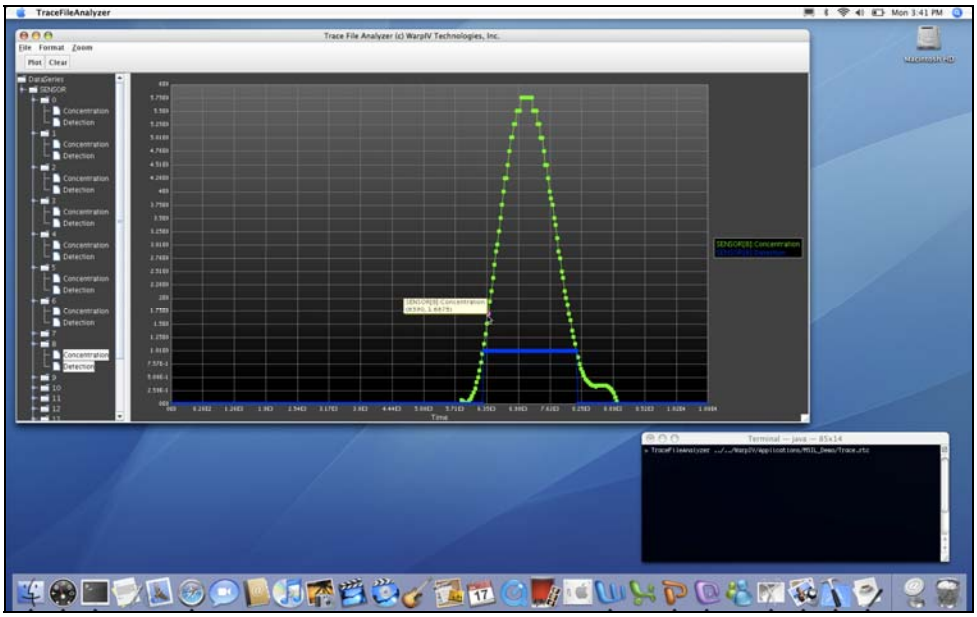

Figure 13: Trace File Analyzer. User-logged data can be automatically be extracted from WarpIV Kernel trace files and plotted over time. Logged data is organized by simulation object type and displayed in a selectable tree-like structure.

To aid in visualizing StatAlgebra distributions, a prototype Blackboard client/server utility was developed to allow any C++ program to pass the distributions to the Java plotter. This requires the WpServer to be running before the C++ program is executed. The Java plotter contains a client that connects to the WpServer and checks for updates sent to the server. The Java plotter can be initiated at any time. Code Segment 13 demonstrates usage of the Blackboard prototype. To add or remove a StatAlgebra distribution from the blackboard, the methods `AddToBlackboard()` and `RemoveFromBlackboard()` simply requires the hostname and port number of the server, and a workspace Id.

```
#include "WpStatDouble.H"
#include "WpStatGaussian.H"
#include “WpSleep.H”

int main() {
  WpStatGaussian gaussian(0.0, 10.0, 10, 5.0, 1.0, "Gaussian"); // Create a Gaussian distribution
  gaussian.AddToBlackboard("localhost", 8001, 1);               // Add it to the blackboard

  WpSleep(10.0);                                                // Sleep 10 seconds

  gaussian.RemoveFromBlackboard("localhost", 8001, 1);          // Remove the distribution
  return 0;
}
```

Code Segment 13: Blackboard Usage. A Gaussian distribution is created with a min of 0.0, max of 10.0, 10 bins, mean of 5.0, sigma of 1.0, and name “Gaussian”. The distribution is added to the WpBlackboard residing on the localhost, port 8001, and workspace Id 1. After sleeping 10.0 seconds, the distribution is removed from the blackboard.

Figure 14 illustrates the blackboard prototype in action. The terminal windows on the right represent the WpServer, Java plotter, and C++ test program. The Java GUI enables the user to select which active distributions to plot. The plotter checks for updates as the user clicks on the distribution menu.

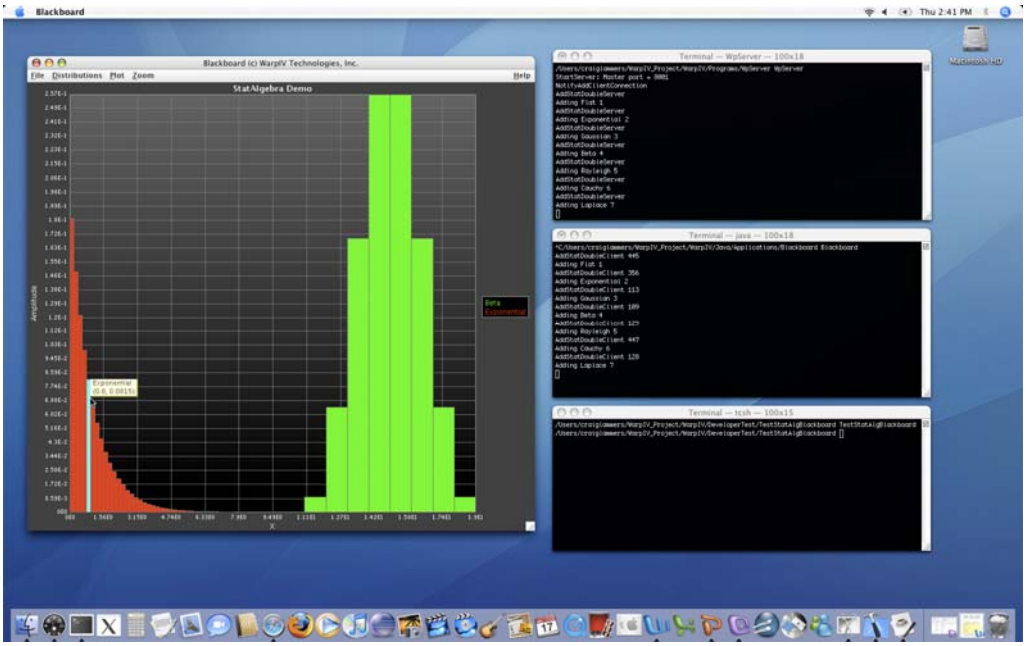

Figure 14: Prototype Distributed Blackboard Utility. This example plots two StatAlgebra histograms. Usage requires executing the following programs: (1) a C++ test application, some of which is shown in Code Segment 13, (2) the Java-based WpBlackboard GUI permitting the user to select the active distributions to display, and (3) the WpServer enabling communication between the WpBlackboard and test application.

7 Comparison with Other Technologies

This section briefly describes existing Course of Action (COA) and M&S technologies along with how they are limited in capability when compared to the approach described in this report.

7.1 Brute Force Monte Carlo Approach

The most common simulation strategy is to run large numbers of Monte Carlo replications to explore different courses of action. The problem with brute force Monte Carlo simulation execution is that many redundant computations are unnecessarily performed. Multi-branching and branch-merging techniques described in this report significantly outperform the brute force Monte Carlo approach. In addition, managing the statistical outputs and accumulating data about branch points and resultant execution paths are simplified because all of the results are now provided by a single execution.

7.2 Simulation Cloning/Forking at Branch Points

The Monte Carlo approach can be extremely wasteful and cumbersome because each replication redundantly computes many of the same calculations as the other replications. One approach to help mitigate this problem is to run a simulation up to the point in time where it makes a branch decision. At that point, the simulation clones or forks itself. If this happens early and often, large numbers of clones will be generated. Cloned simulations do not merge results. In addition, each cloned simulation could still redundantly recompute many of the same events. The branching and branch merging techniques described in this report offer a much more computationally efficient approach by automatically sharing redundant calculations.

7.3 Object Cloning at Branch Points

Another approach has been to clone individual objects within the simulation as branches occur. While this approach is more aggressive than cloning entire simulations, it still suffers from not reusing event computations from redundant branches. The attribute-level branching and merging approaches described in this report provide a much more aggressive and optimal strategy for minimizing redundant computations.

7.4 Grid Computing

Grid computing farms Monte Carlo replications to available compute resources. A more effective use of computing resources would be to efficiently run a multi-branching simulation execution in parallel that does not redundantly compute branches that could be shared between replications. A single laptop with dual-core processors programmed correctly with branching and merging capabilities could potentially outperform a large farm of computing resources managing the execution of brute force Monte Carlo runs.

7.5 Dynamic Assessment and Prediction (DSAP) Tool

DSAP manages large numbers of Monte Carlo replications, and then potentially prunes and restarts them if new data arrives that is not consistent with their simulated state [11]. This approach has some utility in supporting legacy simulations. However it is far from optimal because simulation replications almost always diverge from the real world rapidly. In addition, there is no potential for sharing computations between replications.

7.6 Recursive Simulation

One approach used to project multiple potential outcomes is the notion of recursive simulation (i.e., simulations that use embedded simulation in their event processing) [7]. Recursive simulations do not share computations during branching and they do not merge branches. Therefore, recursive simulation is not as efficient as the HyperWarpSpeed time management algorithm described in this report.

7.7 Integrating Live Data into the Simulation

Traditional approaches collect live data and periodically run simulation executions to predict the future. Each new simulation execution may reproduce a significant amount of redundant computations from previous runs. The strategy described in this report continually feeds live and potentially noisy data into the simulation using control theory such as Kalman Filters to continually calibrate the state. This technique then continually maintains

predictions with multiple branches using the best state estimate of the real world. One additional benefit is that uncertainties from covariance matrices can potentially be extrapolated to characterize the believability of future states. Such error propagation techniques can provide significant value to Verification, Validation, and Accreditation (VV&A) efforts [12]. Again, traditional approaches do not provide this capability.

8 Computational Performance Results

Several initial test applications have been developed for the HyperWarpSpeed algorithm operating within the WarpIV Kernel. Three preliminary test programs are described in the following subsections. These tests collectively (1) verify the correct operation of multi-replication integer, double, and Boolean data types, (2) exercise branching, event splitting, and event merging operations, (3) measure performance in comparison to the equivalent number of Monte Carlo replications, and (4) determine speedup when running on parallel processing architectures. Each test successfully verified the correctness of the multireplication state variable data types, demonstrated repeatable event processing on one or more processors, and highly stressed the branch/splitting/merging algorithms. All of the tests demonstrated significant performance gains over the brute force Monte Carlo approach and obtained substantial parallel speedup when executing on parallel computing systems. Three unique system configurations, defined in Table 2, were used for the following tests.

Table 2: Test System Configurations.

Machine	Processors Specifications	Operating System
Mac Pro	Two Dual-Core Xeon at 3 GHz (total of 4 processors)	Mac OS X 10.4.10
HP Superdome	550 MHz PA8600 (total of 48 processors)	HP-UX 11i
IBM p690	Power4 at 1.3 GHz (total of 32 processors)	Suse Linux Enterprise Server (SLES-10)

8.1 Random Walk

The *Random Walk* test modeled 10,000 entities taking random steps in three dimensions (see Figure 15) using a 128-replication set. To keep the dispersion from spreading too large, the random walk test code was modified to increase the probability of stepping back towards the center.

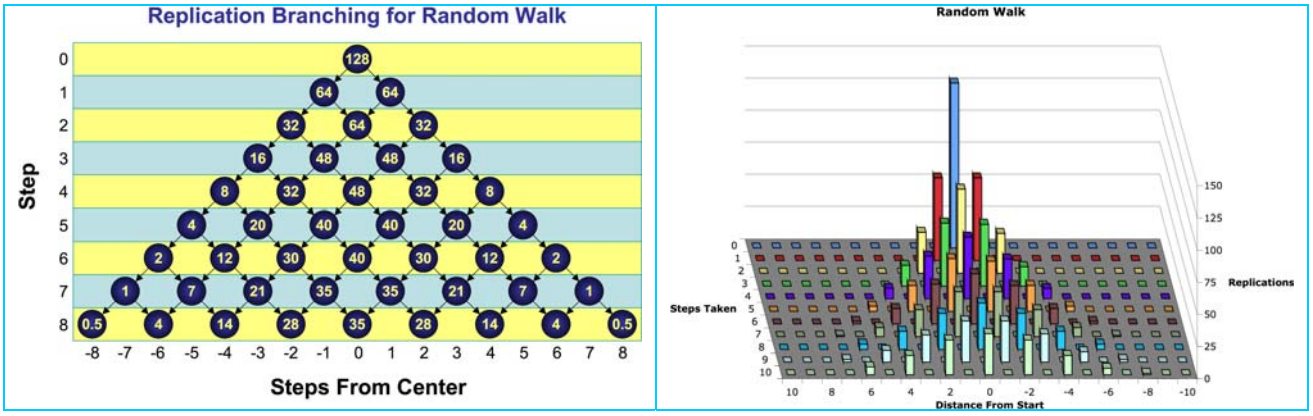

Figure 15: Results from the Random Walk test. All 128 replications start at zero. Each time step performs a branch that takes a step to the left or to the right with an equal probability. The left figure shows how the branching should occur. The right figure plots data taken from an actual run with ten steps.

Random Walk Performance: Mac Pro

The simulation was first executed on the Mac Pro using the brute force Monte Carlo approach. Each replication took 6.02 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 85.15 seconds to complete on a single processor and 26.85 seconds to complete when running in parallel on four processors. The sequential performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 9.05. An additional speedup factor of 3.17 was achieved by executing in parallel on four processors.

Random Walk Performance: HP Superdome

The simulation was then executed on the HP Superdome using the brute force Monte Carlo approach. Each replication took 37.63 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 560.52 seconds to complete on a single processor and 13.98 seconds to complete when running in parallel on forty processors. The sequential performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 8.59. An additional speedup factor of 40.01 was achieved by executing in parallel on forty processors.

Random Walk Performance: IBM p690

The simulation was finally executed on the IBM p690 using the brute force Monte Carlo approach. Each replication took 8.81 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 121.13 seconds to complete on a single processor and 4.75 seconds to complete when running in parallel on thirty processors. The sequential performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 9.31. An additional speedup factor of 25.50 was achieved by executing in parallel on thirty processors.

The random walk test highly stressed the algorithms because every event branched, merged, and changed local variables, while performing almost no computations. The fact that HyperWarpSpeed provided performance improvement over the brute force Monte Carlo approach is notable.

8.2 Aircraft Bombing Target

The *Aircraft Bombing Target* test was developed that flew 1,000 aircraft bombing 1,000 ground targets in sequence. Each aircraft flew in a single-file formation over the same sequence of targets and dropped a bomb on the target if it had not already been destroyed. If the dropped bomb did not destroy the target, a surface to air missile was fired back at the aircraft. Again, a 128-replication set was used in the HyperWarpSpeed algorithm to support this test.

In this scenario, branching occurred at two places (see Figure 16): (1) to determine if the bomb destroyed the target, and (2) to determine if the surface-to-air missile destroyed the aircraft. In both branching cases, a probability of 0.5 was used to determine the effect of (1) the bomb on the target, and (2) the missile on the aircraft.

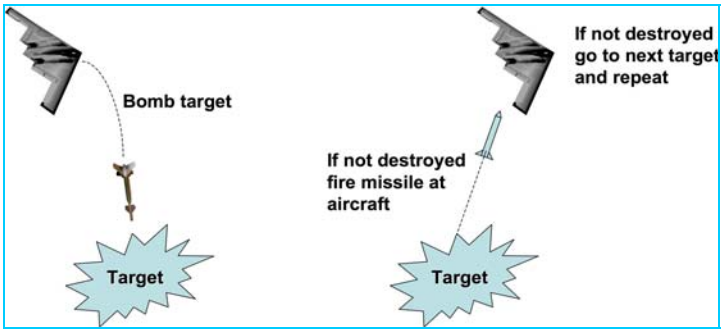

Figure 16: Aircraft bombing target scenario. 1,000 Aircraft bomb 1,000 targets in sequence. If the bomb did not destroy the target, the target fires a surface to air missile back at the aircraft. If the aircraft is not destroyed, then the aircraft moves on to the next target where the sequence of events repeats.

Aircraft Bombing Target Performance: Mac Pro

The simulation was first executed on the Mac Pro using the brute force Monte Carlo approach. Each replication took 10.97 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 33.92 seconds to complete on a single processor and 10.45 seconds to complete when running in parallel on four processors. The performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 41.4. An additional speedup factor of 3.25 was achieved by executing in parallel on four processors.

Aircraft Bombing Target Performance: HP Superdome

The simulation was then executed on the HP Superdome using the brute force Monte Carlo approach. Each replication took 76.27 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 239.24 seconds to complete on a single processor and 6.18 seconds to complete when running in parallel on forty processors. The sequential performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 40.81. An additional speedup factor of 38.71 was achieved by executing in parallel on forty processors.

Aircraft Bombing Target Performance: IBM p690

The simulation was finally executed on the IBM p690 using the brute force Monte Carlo approach. Each replication took 19.66 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 49.39 seconds to complete on a single processor and 2.04 seconds to complete when running in parallel on thirty processors. The sequential performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 50.95. An additional speedup factor of 24.21 was achieved by executing in parallel on thirty processors.

8.3 Miscellaneous Branching Stress Test

The *Miscellaneous Branching Stress Test* was developed that exercised the branching, event-splitting, and event-merging features of the HyperWarpSpeed time management algorithm. Events branched and later merged within objects of type A while other objects of type B queried multi-replication state variables for type A objects. Spin loops burning CPU cycles were added to the models in order to mimic computational workloads representing the actual event processing that might be performed by more realistic applications.

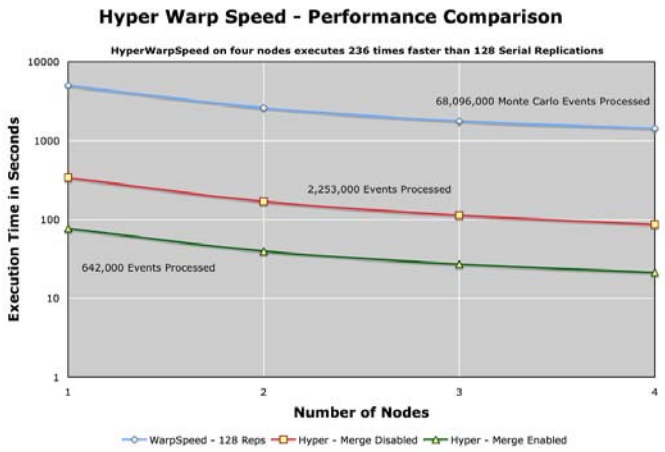

Figure 17: Performance test of the miscellaneous branching stress test executing on the four-processor Mac Pro. For these measurements, the spin loop was increased by a factor of 100. The simulation ran for 100 simulated seconds. This test highly stressed branching, event splitting, and event merging between different types of objects distributed across four processors.

Significant performance improvements for the HyperWarpSpeed algorithm over brute force Monte Carlo replications were observed (see Figure 17). 128 individual Monte Carlo replications would have required the processing of 68,096,000 events. Executing the simulation using HyperWarpSpeed without the branch merging optimization enabled only required 2,253,000 events to be processed. With branch merging enabled, only 642,000 events were actually required to be processed. The results of a single HyperWarpSpeed execution versus 128 independent Monte Carlo replications are statistically equivalent. On four nodes, the HyperWarpSpeed approach ran 236 times faster than 128 serial replications.

Miscellaneous Branching Stress Test Performance: Mac Pro

The simulation was first executed on the Mac Pro for 1000 simulated seconds using the brute force Monte Carlo approach. Each replication took 29.41 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 88.47 seconds to

complete on a single processor and 24.89 seconds to complete when running in parallel on four processors. The performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 42.55. An additional speedup factor of 3.55 was achieved by executing in parallel on four processors.

Miscellaneous Branching Stress Test Performance: HP Superdome

The simulation was then executed for 100 simulated seconds on the HP Superdome using the brute force Monte Carlo approach. Each replication took 257.40 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 493.77 seconds to complete on a single processor and 14.65 seconds to complete when running in parallel on forty processors. The sequential performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 66.73. An additional speedup factor of 33.70 was achieved by executing in parallel on forty processors.

Miscellaneous Branching Stress Test Performance: IBM p690

The simulation was finally executed on the IBM p690 using the brute force Monte Carlo approach. Each replication took 114.40 seconds to complete. The simulation then ran using the HyperWarpSpeed algorithm configured to support the equivalent of 128 replications. HyperWarpSpeed took 222.25 seconds to complete on a single processor and 8.64 seconds to complete when running in parallel on thirty processors. The sequential performance gain of HyperWarpSpeed over 128 independent Monte Carlo runs on a single processor was a factor of 65.89. An additional speedup factor of 25.73 was achieved by executing in parallel on thirty processors.

8.4 Summary of Performance Results

Performance for each of the three tests executing on the three parallel processing hardware systems described above are summarized in Table 3. In all cases, the parallel performance of HyperWarpSpeed offers orders of magnitude speedup gains over the more traditional brute force Monte Carlo approach executed in a serial manner. While these results are preliminary, and do not necessarily indicate high performance for all possible military scenarios, the results are extremely positive and warrant further serious investigation. Obtaining similar performance on realistic military scenarios may require non-traditional conceptual design approaches for constructing models that best take advantage of: branching, merging, support for real-time estimation and prediction, and emerging scalable multicore computing architectures.

Table 3: Summary of HyperWarpSpeed performance for each of the three tests (Random Walk, Aircraft Bombing Mission, and Miscellaneous Branching Test) on three hardware systems (Mac Pro, HP Superdome, and IBM p690).

Performance Random Walk	Single MC Rep	Equivalent 128 MC Reps	Sequential HWS	Parallel HWS	Parallel HWS Speedup	Seq Speedup Over MC Reps	Parallel Speedup Over MC Reps
Mac Pro (4 Nodes)	6.02	770.56	85.15	26.85	3.17	9.05	28.70
HP Superdome (40 Nodes)	37.63	4,816.64	560.52	13.98	40.09	8.59	344.54
IBM p690 (30 Nodes)	8.81	1,127.68	121.13	4.75	25.50	9.31	237.41

Performance Aircraft Bombing Mission	Single MC Rep	Equivalent 128 MC Reps	Sequential HWS	Parallel HWS	Parallel HWS Speedup	Seq Speedup Over MC Reps	Parallel Speedup Over MC Reps
Mac Pro (4 Nodes)	10.97	1,404.16	33.92	10.45	3.25	41.40	134.37
HP Superdome (40 Nodes)	76.27	9,762.56	239.24	6.18	38.71	40.81	1,579.70
IBM p690 (30 Nodes)	19.66	2,516.48	49.39	2.04	24.21	50.95	1,233.57

Performance Miscellaneous Branching Test	Single MC Rep	Equivalent 128 MC Reps	Sequential HWS	Parallel HWS	Parallel HWS Speedup	Seq Speedup Over MC Reps	Parallel Speedup Over MC Reps
Mac Pro (4 Nodes)	29.41	3,764.48	88.47	24.89	3.55	42.55	151.24
HP Superdome (40 Nodes)	257.40	32,947.20	493.77	14.65	33.70	66.73	2,248.96
IBM p690 (30 Nodes)	114.40	14,643.20	222.25	8.64	25.72	65.89	1,694.81

9 Areas of Future Research

To further simplify the use of this technology and to optimize run-time performance, future research and development goals for the HyperWarpSpeed time management algorithm have been identified in the following five general areas.

Goal #1: Extend programming constructs and data types currently supported by other algorithms within the WarpIV Kernel to provide transparent model development.

Goal #2: Extend existing publish/subscribe data distribution mechanisms within the WarpIV Kernel to transparently support branching capabilities.

Goal #3: Research and develop efficient techniques for representing models that branch within the HyperWarpSpeed algorithm.

Goal #4: Develop semi-realistic models of DoD battlefield entities that will verify and validate the correct operation of the HyperWarpSpeed algorithm. These models should be extensible to eventually support more realistic behaviors and scenarios.

Goal #5: Identify potential bottlenecks, optimize algorithms, measure computational improvements over brute force Monte Carlo approaches, and demonstrate scalable performance on a wide range of parallel and distributed computing architectures.

10 Summary and Conclusions

This effort resulted in the development of an efficient and scalable breakthrough M&S technology enabling multi-hypothesis branching, the ability to simulate in the fifth dimension, within a single execution. The aggressive and original approach outlined in this report provides superior improvement over past approaches by managing branches at the state-attribute level, as opposed to cloning objects or simulations at branch points (or worse, executing and managing multiple independent Monte Carlo runs). Memory could easily be exhausted with traditional approaches as branching spirals out of control. In addition to running on parallel and distributed computing architectures, our approach to the problem maximizes computation sharing between branches by enabling the merging of events for branches that are identical at points in time. Further, new estimation and prediction techniques were developed on this effort for calibrating a faster-than-real-time simulation with noisy real-time data using Kalman Filters.

Even though the initial results of the work on this effort were extremely promising, it should be recognized that the technology is still very new. It will take some further experimentation to determine optimal strategies for developing models. The primary concerns are: (1) branching must be done cautiously to minimize unnecessary computations, (2) minimizing the bookkeeping overheads that are required to keep replication sets consistent when modifying multi-replication state variables, and (3) supporting additional modeling constructs such as distributed objects that are published and subscribed in a parallel and distributed processing environment.

All of the aforementioned technology is included in the open source WarpIV Kernel software library. The WarpIV Kernel provides scalable support for parallel and distributed computing and will readily take advantage of next-generation multi-core processors. The WarpIV Kernel is currently the reference implementation for a new Simulation Interoperability Standards Organization (SISO) study group known as the Open Source Initiative for Parallel and Distributed Modeling and Simulation (OSI-PDMS), focusing on simulation architecture and plug-and-play model composability standards.

11 Biographies

JEFFREY S. STEINMAN, President and CEO of WarpIV Technologies, Inc. received his Ph.D. in High Energy Physics from UCLA in 1988 where he studied the quark structure function of high-energy virtual photons at the Stanford Linear Accelerator Center. Upon completing his Ph.D. dissertation, Dr. Steinman worked at the Jet Propulsion Laboratory/CalTech in the area of parallel and distributed computing technologies on hypercube supercomputers. In 1990, Dr. Steinman developed the Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) framework as the next-generation replacement for the Time Warp Operating System (TWOS). This work transitioned to industry in 1996 when Dr. Steinman left JPL/Caltech to join the staff at Metron, Inc. to form its High Performance Computing Division. SPEEDES eventually transitioned into several mainstream programs including BMDS SIM, JMASS, JSIMS, EADTB, and several other smaller and/or R&D programs. In 2000, Dr. Steinman left Metron to join RAM Laboratories as Chief Scientist where he directed all of their high performance computing R&D programs. Dr. Steinman left RAM Laboratories in 2005 to start WarpIV Technologies, Inc., where his focus has been the development of new technologies for high performance computing. Dr. Steinman is the principle developer of the WarpIV Kernel.

CRAIG N. LAMMERS joined WarpIV Technologies in October 2006. As Senior Analyst, his primary contributions have been in the areas of software engineering, development of commercial products, Verification Validation and Accreditation (VV&A) for biological defense, development of tracking algorithms, multiplatform graphical analysis tools using Java, and R&D in estimation and prediction technologies. Craig Lammers has five years of experience as a developer, analyst, and user of the WarpIV Kernel. Prior to WarpIV Technologies, Craig Lammers was a software engineer at RAM Laboratories from 2002 - 2006. In that capacity, he designed, developed, tested, and documented several software applications for AFRL and MDA. Craig Lammers served as principle investigator and developer of the *Detego* neural network for AFRL, was the lead developer of the DSAP multi-replication framework developed for AFRL, and was the principle developer of the WarpIV WanSim wide area network simulation tool for Missile Defense. He has authored over 10 papers on simulation. He graduated with high honors in 2001 from the University of Michigan, Dearborn with a B.S. in Industrial & Systems Engineering.

12 Software Metrics

As a result of the effort, approximately 15,369 lines of C++ and Java code were developed. A breakdown by task is given in Table 4 below. These metrics are conservative in that they do not include modifications/bug fixes to dependent WarpIV utilities uncovered during testing. Development time for most tasks was minimized since the effort built upon and leveraged useful WarpIV Kernel utilities and functionality. In addition to software development, approximately 450 Power Point slides were developed for the WarpIV Kernel training session conducted for AFRL June 4–8, 2007. This training material, along with the software developed during the effort, is included in the WarpIV Kernel distribution for all users of the technology.

Table 4: Software development metrics.

	Software Development (# lines)		
Task	**Source Code**	**Test Code**	**Total**
Multi-Hypothesis Branching			
Multi-Variable Data Type	1368	978	2346
HyperWarpSpeed	782	1371	2153
Real-Time Estimation and Prediction			
Kalman Filters	1380	115	1495
Detection Class	184		184
Matrix Algebra Utility	1664	150	1814
Aircraft Demonstration		2731	2731
Visualization Demonstration		242	242
Statistical Algebra			
Correlations	415	57	472
Multi-Variable Distribution Plotting	219	61	280
Visualization Demonstration		358	358
Measures of Effectiveness and Performance			
Supporting Framework	205		205
Data Logging Utility	386		386
Trace File Analysis	857	801	1658
Visualization Demonstration		384	
Blackboard Prototype	669	57	726
Miscellaneous			
WarpIV Kernel Automatic Testing Framework	319		319
		TOTAL	***15369***

Table 5 lists the developer tests written to (1) verify and validate the correctness of the technology, and (2) conduct benchmarks and performance tests. The developer tests are organized in the table according to the task in which they apply. These tests are located in the `DeveloperTest` directory of the WarpIV Kernel.

Table 5: Developer tests.

Task	Developer Test
Multi-Hypothesis Branching	TestMultiVar
	TestBranchAirMission
	TestBranching
	TestBranching2
	TestRandomWalk
	TestRandomWalker
Real-Time Estimation and Prediction	TestKalmanFilter
	TestMatrix
	TestTracking
Statistical Algebra	TestStatAlgebra
	TestStatAlgBlackboard
	TestMultiVarHist
Measures of Effectiveness and Performance	TestTracking
	TestTrace

13 References

1. Bell B., Santos E. Jr., and Brown S., 2002. “Making Adversary Decision Modeling Tractable with Intent Inference and Information Fusion.” In proceedings of the 11th Conference on Computer Generated Forces and Behavioral Representation.
2. Busch T., "Modeling of Air Operations for Course of Action Determination" in *Enabling Technologies for Simulation Science VI*, Alex Sisti and Dawn Trevisani Editors, Proceedings of SPIE Vol. 4716, pp 35-40, (2002).
3. Chen D., et. al., 2004. “Incremental HLA-Based Distributed Simulation Cloning.” In proceedings of the *2004 Winter Simulation Conference*, pages 386-394.
4. Davis P., and Anderson R., 2004. “Improving the Composability of Department of Defense Models and Simulations.” Prepared for the Defense Modeling and Simulation Office (DMSO). RAND National Defense Research Institute.
5. Fujimoto R., 2000. “Parallel and Distributed Simulation Systems.” John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012.
6. Gelb A., 1974. “Applied Optimal Estimation.” MIT Press.
7. Gilmer, J. B. Jr. and Sullivan F. J., (2005). “Issues in Event Analysis for Recursive Simulation.” In proceedings of the *2005 Winter Simulation Conference*: 8.
8. Hybinette, M. and R. M. Fujimoto. 2001. Cloning parallel simulations. ACM Transactions on Modeling and Computer Simulation, 11: 378-407.
9. Jefferson D., 1985. “Virtual Time.” ACM Transactions on Programming Languages and Systems, Vol. 7, No. 3, pages 404-425.
10. Katsuhiko O., 2005. “Discrete-Time Control Systems (2nd Edition).” ISBN/UPC: 0130342815. Pearson Education.
11. Lammers C., et. al., 2005. “Applying a multi-replication framework to support dynamic situation assessment and predictive capabilities.” Proc. SPIE Vol. 5805, p. 257-268, *Enabling Technologies for Simulation Science IX*; Dawn A. Trevisani, Alex F. Sisti; Editors.
12. Pace, D. K., “Verification, Validation, and Accreditation (VV&A),” In Applied Modeling and Simulation: An Integrated Approach to Development and Operations, D. J. Cloud and L. B. Rainey, editors. Pp. 369-410, McGraw-Hill, New York (1998).
13. Phister P., Busch T., and Plonisch I., 2003. “Joint synthetic battlespace: Cornerstone for predictive battlespace awareness.” In Proceedings of the *Eighth International Command and Control Research and Technology Symposium*.
14. Rothenberg, J., Rand. “A Discussion of Data Quality for Verification, Validation, and Certification (VV&C) of Data to be used in Modeling.” Rand Project Memorandum PM-709-DMSO, Rand, August 1997.
15. Simulation Interoperability Standards Organization (SISO), http://www.sisostds.org.
16. SISO, Open Source Initiative for Parallel and Distributed Modeling and Simulation (OSI-PDMS) study group. http://www.sisostds.org/index.php?tg=articles&idx=More&topics=120&article=471.
17. Steinman J., 1993. "Incremental State Saving in SPEEDES Using C++." In proceedings of the *1993 Winter Simulation Conference*. Pages 687-696.
18. Steinman J. and Hardy D., 2004. “Evolution of the Standard Simulation Architecture.” In proceedings of the *Command and Control Research Technology Symposium*, paper 067.
19. Steinman J, 2005. “The WarpIV Simulation Kernel.” In proceedings of the 2005 *Principles of Advanced and Distributed Simulation (PADS) workshop*.
20. Steinman J. and Busch T., 2006. “Real Time Estimation and Prediction Using Optimistic Simulation and Control Theory Techniques.” In proceedings of the *Spring 2006 Simulation Interoperability Workshop*, 06S-SIW-017.
21. Steinman J., et. al., 2007. “A Proposed Open System Architecture for Modeling and Simulation.” In proceedings of the *Fall 2007 Simulation Interoperability Workshop*, 07F-SIW-044.
22. WarpIV Technologies, Inc., Copyright © 2007, “Statistical Algebra Package - Programming Guide: Version 1.5.”
23. Zarchan P., and Musoff H., 2005. “Fundamentals of Kalman Filtering: A Practical Approach.” AIAA.